Table of Contents

Chapter 1. Modules .............................................................................................................. 1

1.1. FP8 Intrinsics................................................................................................................. 2

FP8 Conversion and Data Movement..................................................................................... 2
C++ struct for handling fp8 data type of e5m2 kind............................................................... 2
C++ struct for handling vector type of two fp8 values of e5m2 kind...................................... 2
C++ struct for handling vector type of four fp8 values of e5m2 kind................................... 3
C++ struct for handling fp8 data type of e4m3 kind............................................................... 3
C++ struct for handling vector type of two fp8 values of e4m3 kind.................................... 3
C++ struct for handling vector type of four fp8 values of e4m3 kind.................................. 3

1.1.1. FP8 Conversion and Data Movement............................................................................ 3

__nv_fp8_interpretation_t...................................................................................................... 3
__nv_saturation_t.................................................................................................................. 3
__nv_fp8_storage_t................................................................................................................ 4
__nv_fp8x2_storage_t............................................................................................................. 4
__nv_fp8x4_storage_t............................................................................................................. 4
__nv_cvt_bfloat16raw2_to_fp8x2.......................................................................................... 4
__nv_cvt_bfloat16raw_to_fp8................................................................................................ 4
__nv_cvt_double2_to_fp8x2.................................................................................................... 5
__nv_cvt_double_to_fp8......................................................................................................... 5
__nv_cvt_float2_to_fp8x2....................................................................................................... 6
__nv_cvt_float_to_fp8........................................................................................................... 6
__nv_cvt_fp8_to_halfraw....................................................................................................... 7
__nv_cvt_fp8x2_to_halfraw2................................................................................................. 7
__nv_cvt_halfraw2_to_fp8x2.................................................................................................. 7
__nv_cvt_halfraw_to_fp8....................................................................................................... 8

1.1.2. C++ struct for handling fp8 data type of e5m2 kind...................................................... 8

__nv_fp8_e5m2..................................................................................................................... 8
__nv_fp8_e5m2::__x............................................................................................................... 8
__nv_fp8_e5m2::__nv_fp8_e5m2............................................................................................ 8
__nv_fp8_e5m2::__nv_fp8_e5m2............................................................................................ 9
__nv_fp8_e5m2::__nv_fp8_e5m2............................................................................................ 9
__nv_fp8_e5m2::__nv_fp8_e5m2............................................................................................ 9
__nv_fp8_e5m2::__nv_fp8_e5m2............................................................................................ 9
__nv_fp8_e5m2::__nv_fp8_e5m2............................................................................................ 9
__nv_fp8_e5m2::__nv_fp8_e5m2............................................................................................ 10
__nv_fp8_e5m2::__nv_fp8_e5m2............................................................................................ 10
1.1.5. C++ struct for handling fp8 data type of e4m3 kind......................................................... 17
__nv_fp8_e4m3........................................................................................................................17
__nv_fp8_e4m3::__x................................................................................................................17
__nv_fp8_e4m3::__nv_fp8_e4m3............................................................................................17
__nv_fp8_e4m3::__nv_fp8_e4m3............................................................................................18
__nv_fp8_e4m3::__nv_fp8_e4m3............................................................................................18
__nv_fp8_e4m3::__nv_fp8_e4m3............................................................................................18
__nv_fp8_e4m3::__nv_fp8_e4m3............................................................................................18
__nv_fp8_e4m3::__nv_fp8_e4m3............................................................................................18
__nv_fp8_e4m3::__nv_fp8_e4m3............................................................................................19
__nv_fp8_e4m3::__nv_fp8_e4m3............................................................................................19
__nv_fp8_e4m3::__nv_fp8_e4m3............................................................................................19
__nv_fp8_e4m3::__nv_fp8_e4m3............................................................................................19
__nv_fp8_e4m3::__nv_fp8_e4m3............................................................................................19
__nv_fp8_e4m3::__nv_fp8_e4m3............................................................................................19
__nv_fp8_e4m3::__nv_fp8_e4m3............................................................................................20
__nv_fp8_e4m3::operator __half............................................................................................20
__nv_fp8_e4m3::operator __nv_bfloat16............................................................................... 20
__nv_fp8_e4m3::operator bool...............................................................................................20
__nv_fp8_e4m3::operator char.............................................................................................. 20
__nv_fp8_e4m3::operator double...........................................................................................20
__nv_fp8_e4m3::operator float.............................................................................................. 21
__nv_fp8_e4m3::operator int..................................................................................................21
__nv_fp8_e4m3::operator long int.........................................................................................21
__nv_fp8_e4m3::operator long long int................................................................................21
__nv_fp8_e4m3::operator short int......................................................................................21
__nv_fp8_e4m3::operator signed char.................................................................................. 21
__nv_fp8_e4m3::operator unsigned char................................................................................. 22
__nv_fp8_e4m3::operator unsigned int.................................................................................. 22
__nv_fp8_e4m3::operator unsigned long int........................................................................... 22
__nv_fp8_e4m3::operator unsigned long long int..................................................................... 22
__nv_fp8_e4m3::operator unsigned short int.......................................................................... 22
1.1.6. C++ struct for handling vector type of two fp8 values of e4m3 kind.............................. 23
__nv_fp8x2_e4m3.................................................................................................................... 23
__nv_fp8x2_e4m3::__x............................................................................................................ 23
__nv_fp8x2_e4m3::__nv_fp8x2_e4m3.................................................................................... 23
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>__nv_fp8x2_e4m3::__nv_fp8x2_e4m3</td>
<td>23</td>
</tr>
<tr>
<td>__nv_fp8x2_e4m3::__nv_fp8x2_e4m3</td>
<td>23</td>
</tr>
<tr>
<td>__nv_fp8x2_e4m3::__nv_fp8x2_e4m3</td>
<td>24</td>
</tr>
<tr>
<td>__nv_fp8x2_e4m3::__nv_fp8x2_e4m3</td>
<td>24</td>
</tr>
<tr>
<td>__nv_fp8x2_e4m3::operator __half2</td>
<td>24</td>
</tr>
<tr>
<td>__nv_fp8x2_e4m3::operator float2</td>
<td>24</td>
</tr>
</tbody>
</table>

1.1.7. C++ struct for handling vector type of four fp8 values of e4m3 kind

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>__nv_fp8x4_e4m3</td>
<td>24</td>
</tr>
<tr>
<td>__nv_fp8x4_e4m3::__x</td>
<td>24</td>
</tr>
<tr>
<td>__nv_fp8x4_e4m3::__nv_fp8x4_e4m3</td>
<td>25</td>
</tr>
<tr>
<td>__nv_fp8x4_e4m3::__nv_fp8x4_e4m3</td>
<td>25</td>
</tr>
<tr>
<td>__nv_fp8x4_e4m3::__nv_fp8x4_e4m3</td>
<td>25</td>
</tr>
<tr>
<td>__nv_fp8x4_e4m3::__nv_fp8x4_e4m3</td>
<td>25</td>
</tr>
<tr>
<td>__nv_fp8x4_e4m3::__nv_fp8x4_e4m3</td>
<td>25</td>
</tr>
<tr>
<td>__nv_fp8x4_e4m3::__nv_fp8x4_e4m3::operator float4</td>
<td>25</td>
</tr>
</tbody>
</table>

1.2. Half Precision Intrinsics

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>__half</td>
<td>26</td>
</tr>
<tr>
<td>__half2</td>
<td>26</td>
</tr>
<tr>
<td>__half2_raw</td>
<td>26</td>
</tr>
<tr>
<td>__half2_raw..</td>
<td>26</td>
</tr>
<tr>
<td>Half Arithmetic Constants</td>
<td>26</td>
</tr>
<tr>
<td>Half Arithmetic Functions</td>
<td>27</td>
</tr>
<tr>
<td>Half2 Arithmetic Functions</td>
<td>27</td>
</tr>
<tr>
<td>Half Comparison Functions</td>
<td>27</td>
</tr>
<tr>
<td>Half2 Comparison Functions</td>
<td>27</td>
</tr>
<tr>
<td>Half Precision Conversion and Data Movement</td>
<td>27</td>
</tr>
<tr>
<td>Half Math Functions</td>
<td>27</td>
</tr>
<tr>
<td>Half2 Math Functions</td>
<td>27</td>
</tr>
<tr>
<td>__nv_half</td>
<td>27</td>
</tr>
<tr>
<td>__nv_half2</td>
<td>27</td>
</tr>
<tr>
<td>__nv_half2 Raw</td>
<td>27</td>
</tr>
<tr>
<td>__nv_half2_raw</td>
<td>27</td>
</tr>
<tr>
<td>half</td>
<td>27</td>
</tr>
<tr>
<td>half2</td>
<td>28</td>
</tr>
<tr>
<td>nv_half</td>
<td>28</td>
</tr>
<tr>
<td>nv_half2</td>
<td>28</td>
</tr>
</tbody>
</table>

1.2.1. Half Arithmetic Constants

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDART_INF_FP16</td>
<td>28</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------</td>
<td>------</td>
</tr>
<tr>
<td>__h2div</td>
<td>37</td>
</tr>
<tr>
<td>__habs2</td>
<td>37</td>
</tr>
<tr>
<td>__hadd2</td>
<td>37</td>
</tr>
<tr>
<td>__hadd2_rn</td>
<td>37</td>
</tr>
<tr>
<td>__hadd2_sat</td>
<td>38</td>
</tr>
<tr>
<td>__hcmadd</td>
<td>38</td>
</tr>
<tr>
<td>__hfma2</td>
<td>39</td>
</tr>
<tr>
<td>__hfma2_relu</td>
<td>39</td>
</tr>
<tr>
<td>__hfma2_sat</td>
<td>39</td>
</tr>
<tr>
<td>__hmul2</td>
<td>40</td>
</tr>
<tr>
<td>__hmul2_rn</td>
<td>40</td>
</tr>
<tr>
<td>__hmul2_sat</td>
<td>40</td>
</tr>
<tr>
<td>__hneg2</td>
<td>41</td>
</tr>
<tr>
<td>__hsub2</td>
<td>41</td>
</tr>
<tr>
<td>__hsub2_rn</td>
<td>41</td>
</tr>
<tr>
<td>__hsub2_sat</td>
<td>42</td>
</tr>
<tr>
<td>atomicAdd</td>
<td>42</td>
</tr>
<tr>
<td>operator*</td>
<td>43</td>
</tr>
<tr>
<td>operator*=</td>
<td>43</td>
</tr>
<tr>
<td>operator+</td>
<td>43</td>
</tr>
<tr>
<td>operator+=</td>
<td>43</td>
</tr>
<tr>
<td>operator++</td>
<td>43</td>
</tr>
<tr>
<td>operator++=</td>
<td>44</td>
</tr>
<tr>
<td>operator-</td>
<td>44</td>
</tr>
<tr>
<td>operator-=</td>
<td>44</td>
</tr>
<tr>
<td>operator--</td>
<td>44</td>
</tr>
<tr>
<td>operator==</td>
<td>44</td>
</tr>
<tr>
<td>operator/=</td>
<td>45</td>
</tr>
<tr>
<td>operator/=</td>
<td>45</td>
</tr>
</tbody>
</table>

1.2.4. Half Comparison Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>__heq</td>
<td>45</td>
</tr>
<tr>
<td>__hequ</td>
<td>46</td>
</tr>
<tr>
<td>__hge</td>
<td>46</td>
</tr>
<tr>
<td>__hgeu</td>
<td>47</td>
</tr>
<tr>
<td>__hgt</td>
<td>47</td>
</tr>
<tr>
<td>__hgtu</td>
<td>48</td>
</tr>
</tbody>
</table>
1.2.5. Half2 Comparison Functions

__hbeq2.................................................................................................................................... 54
__hbequ2.................................................................................................................................. 55
__hbge2.................................................................................................................................... 55
__hbgeu2.................................................................................................................................. 56
__hbgt2.................................................................................................................................... 56
__hbgtu2.................................................................................................................................... 57
__hble2.................................................................................................................................... 58
__hbleu2.................................................................................................................................... 58
__hblt2.................................................................................................................................... 58
__hbltu2.................................................................................................................................... 59
__hbne2.................................................................................................................................... 60
__hbneu2.................................................................................................................................... 61
__heq2.................................................................................................................................... 61
__heq2_mask............................................................................................................................. 62
__hequ2.................................................................................................................................... 62
__hequ2_mask............................................................................................................................ 63
__hge2.................................................................................................................................... 63
__hge2_mask............................................................................................................................. 64
__hgeu2.................................................................................................................................... 64
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>__hgeu2_mask</td>
<td>65</td>
</tr>
<tr>
<td>__hgt2</td>
<td>65</td>
</tr>
<tr>
<td>__hgtu2_mask</td>
<td>66</td>
</tr>
<tr>
<td>__hgtu2</td>
<td>66</td>
</tr>
<tr>
<td>__hisnan2</td>
<td>67</td>
</tr>
<tr>
<td>__hle2</td>
<td>67</td>
</tr>
<tr>
<td>__hle2_mask</td>
<td>67</td>
</tr>
<tr>
<td>__hleu2</td>
<td>68</td>
</tr>
<tr>
<td>__hleu2_mask</td>
<td>68</td>
</tr>
<tr>
<td>__hlt2</td>
<td>69</td>
</tr>
<tr>
<td>__hlt2_mask</td>
<td>69</td>
</tr>
<tr>
<td>__hltu2</td>
<td>70</td>
</tr>
<tr>
<td>__hltu2_mask</td>
<td>70</td>
</tr>
<tr>
<td>__hmax2</td>
<td>71</td>
</tr>
<tr>
<td>__hmax2_nan</td>
<td>71</td>
</tr>
<tr>
<td>__hmin2</td>
<td>72</td>
</tr>
<tr>
<td>__hmin2_nan</td>
<td>72</td>
</tr>
<tr>
<td>__hne2</td>
<td>72</td>
</tr>
<tr>
<td>__hne2_mask</td>
<td>73</td>
</tr>
<tr>
<td>__hneu2</td>
<td>73</td>
</tr>
<tr>
<td>__hneu2_mask</td>
<td>74</td>
</tr>
<tr>
<td>operator!=</td>
<td>75</td>
</tr>
<tr>
<td>operator&lt;</td>
<td>75</td>
</tr>
<tr>
<td>operator&lt;=</td>
<td>75</td>
</tr>
<tr>
<td>operator==</td>
<td>75</td>
</tr>
<tr>
<td>operator&gt;</td>
<td>76</td>
</tr>
<tr>
<td>operator&gt;=</td>
<td>76</td>
</tr>
<tr>
<td>1.2.6. Half Precision Conversion and Data Movement</td>
<td>76</td>
</tr>
<tr>
<td>__double2half</td>
<td>76</td>
</tr>
<tr>
<td>__float2half2_rn</td>
<td>76</td>
</tr>
<tr>
<td>__float2half</td>
<td>77</td>
</tr>
<tr>
<td>__float2half2_rn</td>
<td>77</td>
</tr>
<tr>
<td>__float2half_rd</td>
<td>78</td>
</tr>
<tr>
<td>__float2half_fn</td>
<td>78</td>
</tr>
<tr>
<td>__float2half_ru</td>
<td>78</td>
</tr>
<tr>
<td>__float2half_rz</td>
<td>79</td>
</tr>
<tr>
<td>__floats2half2_rn</td>
<td>79</td>
</tr>
</tbody>
</table>
CUDA Math API

__half22float2.......................................................................................................................... 80
__half2char_rz.........................................................................................................................80
__half2float............................................................................................................................ 81
__half2half2............................................................................................................................ 81
__half2int_rd........................................................................................................................... 81
__half2int_rn........................................................................................................................... 82
__half2int_ru........................................................................................................................... 82
__half2int_rz........................................................................................................................... 83
__half2ll_rd.............................................................................................................................. 83
__half2ll_rn.............................................................................................................................. 83
__half2ll_ru.............................................................................................................................. 84
__half2ll_rz.............................................................................................................................. 84
__half2short_rd....................................................................................................................... 85
__half2short_rn....................................................................................................................... 85
__half2short_ru....................................................................................................................... 86
__half2short_rz....................................................................................................................... 86
__half2uchar_rz.......................................................................................................................86
__half2uint_rd......................................................................................................................... 87
__half2uint_rn......................................................................................................................... 87
__half2uint_ru......................................................................................................................... 88
__half2uint_rz......................................................................................................................... 88
__half2ull_rd............................................................................................................................88
__half2ull_rn............................................................................................................................89
__half2ull_ru............................................................................................................................89
__half2ull_rz............................................................................................................................ 90
__half2ushort_rd..................................................................................................................... 90
__half2ushort_rn..................................................................................................................... 91
__half2ushort_ru..................................................................................................................... 91
__half2ushort_rz..................................................................................................................... 92
__half_as_short.......................................................................................................................92
__half_as_ushort.....................................................................................................................93
__halves2half2.........................................................................................................................93
__high2float.............................................................................................................................94
__high2half..............................................................................................................................94
__high2half2.............................................................................................................................94
__highs2half2........................................................................................................................... 95
__int2half_rd..........................................................................................................................95
__int2half_rn.......................................................................................................................... 96
__int2half_ru.................................................................96
__int2half_rz.................................................................96
__ldca..............................................................................97
__ldca..............................................................................97
__ldcg.............................................................................97
__ldcg.............................................................................97
__ldcs.............................................................................98
__ldcs.............................................................................98
__ldcv.............................................................................98
__ldcv.............................................................................98
__ldg................................................................................99
__ldg................................................................................99
__ldlu...............................................................................99
__ldlu...............................................................................99
__ll2half_rd..................................................................100
__ll2half_rn..................................................................100
__ll2half_ru..................................................................101
__ll2half_rz..................................................................101
__low2float...................................................................102
__low2half.....................................................................102
__low2half2...................................................................102
__lowhigh2highlow.........................................................103
__lows2half2...................................................................103
__shfl_down_sync...........................................................104
__shfl_down_sync...........................................................104
__shfl_sync.....................................................................105
__shfl_sync.....................................................................105
__shfl_up_sync...............................................................106
__shfl_up_sync...............................................................106
__shfl_xor_sync..............................................................108
__shfl_xor_sync..............................................................108
__short2half_rd..............................................................110
__short2half_rn..............................................................110
__short2half_ru..............................................................111
__short2half_rz..............................................................111
__short_as_half..............................................................112
__stcg.............................................................................112
__stcg.............................................................................112

CUDA Math API

v12.4 | xi
__stcs .....................................................................................................................................113
__stwb ....................................................................................................................................113
__stwt .....................................................................................................................................114
__uint2half_rd ..........................................................114
__uint2half_rn ..........................................................115
__uint2half_ru ..........................................................115
__uint2half_rz ..........................................................116
__ull2half_rd ................................................................116
__ull2half_rn ................................................................116
__ull2half_ru ................................................................117
__ull2half_rz ................................................................117
__ushort2half_rd ..........................................................118
__ushort2half_rn ..........................................................118
__ushort2half_ru ..........................................................119
__ushort2half_rz ..........................................................119
__ushort_as_half ..........................................................120
make_half2 ..........................................................120

1.2.7. Half Math Functions ..........................................................120
hceil ........................................................................121
hcos ........................................................................121
hexp ........................................................................121
hexp10 .......................................................................122
hexp2 .......................................................................122
hfloor ........................................................................122
hlog ........................................................................123
hlog10 .......................................................................123
hlog2 .......................................................................124
hrcp ........................................................................124
hrint ........................................................................124
hrint ........................................................................124
hrsqrt .......................................................................125
hsin ........................................................................125
hsqrt ........................................................................126
htrunc .......................................................................126

1.2.8. Half2 Math Functions ..........................................................126
h2ceil .......................................................................127
1.3. Bfloat16 Precision Intrinsics ............................................................................................... 132
    __nv_bfloat16............................................................................................................................ 133
    __nv_bfloat162.............................................................................................................................. 134
    CUDART_INF_BF16.................................................................................................................. 134
    CUDART_MAX_NORMAL_BF16................................................................................................. 135
    CUDART_MIN_DENORM_BF16.................................................................................................... 135
    CUDART_NAN_BF16.................................................................................................................. 135
    CUDART_NEG_ZERO_BF16........................................................................................................ 135
    CUDART_ONE_BF16.................................................................................................................. 135
    CUDART_ZERO_BF16.................................................................................................................. 135
    1.3.1. Bfloat16 Arithmetic Constants...................................................................................... 134
    CUDART_INF_BF16.................................................................................................................. 134
    CUDART_MAX_NORMAL_BF16................................................................................................. 135
    CUDART_MIN_DENORM_BF16.................................................................................................... 135
    CUDART_NAN_BF16.................................................................................................................. 135
    CUDART_NEG_ZERO_BF16........................................................................................................ 135
    CUDART_ONE_BF16.................................................................................................................. 135
    CUDART_ZERO_BF16.................................................................................................................. 135
    1.3.2. Bfloat16 Arithmetic Functions...................................................................................... 135
    nv_bfloat16................................................................................................................................ 134
    nv_bfloat162.............................................................................................................................. 134
    Bfloat162 Comparison Functions............................................................................................ 134
    nv_bfloat162.............................................................................................................................. 134
    nv_bfloat162.............................................................................................................................. 134
__h2div................................................................................................................................... 135
__habs....................................................................................................................................136
__hadd....................................................................................................................................136
__hadd_rn................................................................................................................................136
__hadd_sat................................................................................................................................136
__hdiv..................................................................................................................................... 137
__hfma................................................................................................................................... 137
__hfma_relu...........................................................................................................................137
__hfma_sat................................................................................................................................138
__hmul..................................................................................................................................... 138
__hmul_rn................................................................................................................................139
__hmul_sat................................................................................................................................139
__hneg....................................................................................................................................139
__hsub....................................................................................................................................140
__hsub_rn................................................................................................................................140
__hsub_sat................................................................................................................................140
atomicAdd..............................................................................................................................141
operator*................................................................................................................................141
operator*=&................................................................................................................................141
operator+................................................................................................................................142
operator+=&................................................................................................................................142
operator++................................................................................................................................142
operator++=&..........................................................................................................................142
operator-................................................................................................................................142
operator-=&................................................................................................................................142
operator--................................................................................................................................143
operator--=&..........................................................................................................................143
operator-=................................................................................................................................142
operator/.................................................................................................................................. 143
operator/=................................................................................................................................143
1.3.3. Bfloat162 Arithmetic Functions.................................................................................. 144
__habs2...................................................................................................................................144
__hadd2...................................................................................................................................144
__hadd2_rn................................................................................................................................144
__hadd2_sat................................................................................................................................145
__hcmadd................................................................................................................................... 145
__hfma2....................................................................................................................................146
__hfma2_relu.........................................................................................................................146
__hfma2_sat.......................................................................................................................... 146
__hmul2................................................................................................................................. 147
__hmul2_rn............................................................................................................................ 147
__hmul2_sat.......................................................................................................................... 147
__hneg2..................................................................................................................................148
__hsub2..................................................................................................................................148
__hsub2_rn............................................................................................................................ 148
__hsub2_sat...........................................................................................................................149
atomicAdd..............................................................................................................................149
operator*................................................................................................................................150
operator*=..............................................................................................................................150
operator+............................................................................................................................... 150
operator+=............................................................................................................................. 150
operator++............................................................................................................................. 151
operator++=........................................................................................................................... 151
operator-................................................................................................................................151
operator-=..............................................................................................................................151
operator--..............................................................................................................................152
operator//=........................................................................................................................... 152
operator/................................................................................................................................ 152
operator/=.............................................................................................................................. 152
1.3.4. Bfloat16 Comparison Functions...................................................................................152
__heq......................................................................................................................................152
__hequ....................................................................................................................................153
__hge......................................................................................................................................153
__hgeu....................................................................................................................................154
__hgt...................................................................................................................................... 154
__hgtu..................................................................................................................................... 155
__hisinf...................................................................................................................................155
__hisnan.................................................................................................................................156
__hle.......................................................................................................................................156
__hleu.................................................................................................................................... 157
__hlt....................................................................................................................................... 157
__hltu..................................................................................................................................... 158
__hmax...................................................................................................................................158
1.3.6. Bfloat16 Precision Conversion and Data Movement

__bfloat162float2................................................................. 184
__bfloat162bfloat162.......................................................... 184
__bfloat162char_rz................................................................ 185
__bfloat162float................................................................. 185
__bfloat162int_rd................................................................. 186
__bfloat162int_rn................................................................. 186
__bfloat162int_ru................................................................. 187
__bfloat162int_rz................................................................. 187
__bfloat162ll_rd................................................................. 188
__bfloat162ll_rn................................................................. 188
__bfloat162ll_ru................................................................. 189
__bfloat162ll_rz................................................................. 189
__bfloat162short_rd......................................................... 190
__bfloat162short_rn......................................................... 190
__bfloat162short_ru......................................................... 191
__bfloat162short_rz......................................................... 191

__hle2_mask........................................................................ 176
__hleu2............................................................................... 176
__hleu2_mask....................................................................... 177
__hlt2.................................................................................. 177
__hlt2_mask......................................................................... 178
__hltu2.............................................................................. 178
__hltu2_mask....................................................................... 179
__hmax2............................................................................. 179
__hmax2_nan....................................................................... 180
__hmin2............................................................................. 180
__hmin2_nan....................................................................... 180
__hne2.............................................................................. 181
__hne2_mask....................................................................... 181
__hneu2.............................................................................. 182
__hneu2_mask...................................................................... 182
operator!=......................................................................... 183
operator<......................................................................... 183
operator<=......................................................................... 183
operator==......................................................................... 183
operator>......................................................................... 184
operator>==....................................................................... 184
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>__bfloat162uchar_rz</td>
<td>192</td>
</tr>
<tr>
<td>__bfloat162uint_rd</td>
<td>192</td>
</tr>
<tr>
<td>__bfloat162uint_rn</td>
<td>193</td>
</tr>
<tr>
<td>__bfloat162uint_ru</td>
<td>193</td>
</tr>
<tr>
<td>__bfloat162uint_rz</td>
<td>194</td>
</tr>
<tr>
<td>__bfloat162ull_rd</td>
<td>194</td>
</tr>
<tr>
<td>__bfloat162ull_rn</td>
<td>195</td>
</tr>
<tr>
<td>__bfloat162ull_ru</td>
<td>195</td>
</tr>
<tr>
<td>__bfloat162ull_rz</td>
<td>196</td>
</tr>
<tr>
<td>__bfloat162ushort_rd</td>
<td>196</td>
</tr>
<tr>
<td>__bfloat162ushort_rn</td>
<td>197</td>
</tr>
<tr>
<td>__bfloat162ushort_ru</td>
<td>197</td>
</tr>
<tr>
<td>__bfloat162ushort_rz</td>
<td>198</td>
</tr>
<tr>
<td>__bfloat16_as_short</td>
<td>198</td>
</tr>
<tr>
<td>__double2bfloat16</td>
<td>199</td>
</tr>
<tr>
<td>__float2bfloat16_rn</td>
<td>199</td>
</tr>
<tr>
<td>__float2bfloat16</td>
<td>200</td>
</tr>
<tr>
<td>__float2bfloat16_rn</td>
<td>200</td>
</tr>
<tr>
<td>__float2bfloat16_rd</td>
<td>201</td>
</tr>
<tr>
<td>__float2bfloat16_rn</td>
<td>201</td>
</tr>
<tr>
<td>__float2bfloat16_ru</td>
<td>202</td>
</tr>
<tr>
<td>__float2bfloat16_rz</td>
<td>202</td>
</tr>
<tr>
<td>__floats2bfloat16_rn</td>
<td>202</td>
</tr>
<tr>
<td>__halves2bfloat16</td>
<td>203</td>
</tr>
<tr>
<td>__high2bfloat16</td>
<td>203</td>
</tr>
<tr>
<td>__high2bfloat16_rn</td>
<td>204</td>
</tr>
<tr>
<td>__high2float</td>
<td>204</td>
</tr>
<tr>
<td>__highs2bfloat16</td>
<td>205</td>
</tr>
<tr>
<td>__int2bfloat16_rd</td>
<td>205</td>
</tr>
<tr>
<td>__int2bfloat16_rn</td>
<td>206</td>
</tr>
<tr>
<td>__int2bfloat16_ru</td>
<td>206</td>
</tr>
<tr>
<td>__int2bfloat16_rz</td>
<td>206</td>
</tr>
<tr>
<td>__ldca</td>
<td>207</td>
</tr>
<tr>
<td>__ldca</td>
<td>207</td>
</tr>
<tr>
<td>__ldcg</td>
<td>207</td>
</tr>
<tr>
<td>__ldcg</td>
<td>208</td>
</tr>
<tr>
<td>__ldcs</td>
<td>208</td>
</tr>
</tbody>
</table>
__uint2bfloat16_rn................................................................................................................ 225
__uint2bfloat16_ru................................................................................................................ 225
__uint2bfloat16_rz.................................................................................................................226
__ull2bfloat16_rd...................................................................................................................226
__ull2bfloat16_rn...................................................................................................................226
__ull2bfloat16_ru...................................................................................................................227
__ull2bfloat16_rz...................................................................................................................227
__ushort2bfloat16_rd............................................................................................................ 228
__ushort2bfloat16_rn............................................................................................................228
__ushort2bfloat16_ru............................................................................................................228
__ushort2bfloat16_rz............................................................................................................ 229
__ushort_as_bfloat16............................................................................................................229
make_bfloat162..................................................................................................................... 230

1.3.7. Bfloat16 Math Functions...............................................................................................230
hceil........................................................................................................................................230
hcos........................................................................................................................................231
hexp.......................................................................................................................................231
hexp10..................................................................................................................................231
hexp2.....................................................................................................................................231
hfloor......................................................................................................................................232
hlog........................................................................................................................................ 233
hlog10.....................................................................................................................................233
hlog2...................................................................................................................................... 233
hrcp........................................................................................................................................ 234
hrint........................................................................................................................................234
hrsqrt.....................................................................................................................................235
hsin.........................................................................................................................................235
hsqrt.......................................................................................................................................235
htrunc.....................................................................................................................................236

1.3.8. Bfloat162 Math Functions............................................................................................. 236
h2ceil......................................................................................................................................236
h2cos......................................................................................................................................237
h2exp.....................................................................................................................................237
h2exp10..................................................................................................................................238
h2exp2....................................................................................................................................238
h2floor....................................................................................................................................238
h2log...................................................................................................................................... 239
h2log10..................................................................................................................................239
1.4. Mathematical Functions ................................................................. 242
1.5. Single Precision Mathematical Functions ....................................... 243
acosf........................................................................................................ 243
acoshf...................................................................................................... 243
asinf........................................................................................................ 244
asinhf...................................................................................................... 244
atan2f..................................................................................................... 245
atanf....................................................................................................... 246
atanhf..................................................................................................... 246
cbrtf....................................................................................................... 247
ceilf........................................................................................................ 247
copysignf.............................................................................................. 247
cosf........................................................................................................ 248
coshf...................................................................................................... 248
cosf.......................................................... 248
cospif..................................................................................................... 249
cyl_bessel_i0f...................................................................................... 249
cyl_bessel_i1f...................................................................................... 249
erfcf....................................................................................................... 250
erfcinvf................................................................................................. 250
erfcxf..................................................................................................... 251
erff........................................................................................................... 251
erff........................................................................................................... 251
erfinvf.................................................................................................... 252
exp10f................................................................................................... 252
exp2f..................................................................................................... 253
expf........................................................................................................ 253
expm1f.................................................................................................. 254
fabsf...................................................................................................... 254
fdimf..................................................................................................... 255
fdividef.................................................................................................. 255
floorf...................................................................................................... 256
fmaf........................................................................................................ 256
1.6. Double Precision Mathematical Functions

acos.................................................................286
acosh.............................................................286
asin...............................................................287
asinh.............................................................287
atan...............................................................288
atan2.............................................................288
atanh.............................................................289
cbrait..............................................................289
ceil...............................................................290
copysign........................................................290
cos...............................................................290
cosh.............................................................291
cospi..............................................................291
cyl_bessel_i0....................................................292
max............................................................................................................................................ 309
max............................................................................................................................................ 309
min............................................................................................................................................. 310
min............................................................................................................................................. 310
min............................................................................................................................................. 310
modf...........................................................................................................................................310
nan............................................................................................................................................. 311
nearbyint....................................................................................................................................311
nextafter.................................................................................................................................... 312
norm.......................................................................................................................................... 312
norm3d...................................................................................................................................... 313
norm4d...................................................................................................................................... 313
normcdf..................................................................................................................................... 314
normcdfinv.................................................................................................................................314
pow.............................................................................................................................................315
rcbrt........................................................................................................................................... 316
remainder................................................................................................................................... 316
remquo...................................................................................................................................... 317
rhypot......................................................................................................................................... 317
rint.............................................................................................................................................. 318
rnorm.......................................................................................................................................... 318
rnorm3d.....................................................................................................................................319
rnorm4d.....................................................................................................................................319
round.......................................................................................................................................... 320
rsqrt........................................................................................................................................... 320
scalbln....................................................................................................................................... 321
scalbn........................................................................................................................................ 321
signbit........................................................................................................................................ 321
sin.............................................................................................................................................. 322
sincos.........................................................................................................................................322
sincospi......................................................................................................................................323
sinh............................................................................................................................................ 323
sinpi........................................................................................................................................... 324
sqrt.............................................................................................................................................324
tan.............................................................................................................................................. 325
tanh............................................................................................................................................325
tgamma..................................................................................................................................... 325
trunc.......................................................................................................................................... 326
1.7. Integer Mathematical Functions.................................................................................. 328
  abs...................................................................................................................................... 328
  labs.................................................................................................................................... 328
  llabs................................................................................................................................. 328
  llmax................................................................................................................................. 329
  llmin................................................................................................................................. 329
  max................................................................................................................................. 329
  max................................................................................................................................. 330
  max................................................................................................................................. 330
  max................................................................................................................................. 330
  max................................................................................................................................. 330
  max................................................................................................................................. 330
  max................................................................................................................................. 330
  min................................................................................................................................. 332
  min................................................................................................................................. 332
  min................................................................................................................................. 332
  min................................................................................................................................. 333
  min................................................................................................................................. 333
  min................................................................................................................................. 333
  min................................................................................................................................. 333
  min................................................................................................................................. 333
  min................................................................................................................................. 333
  min................................................................................................................................. 334
  min................................................................................................................................. 334
  min................................................................................................................................. 334
  ullmax............................................................................................................................ 334
  ullmin............................................................................................................................ 334
  umax............................................................................................................................... 335
  umin............................................................................................................................... 335

1.8. Single Precision Intrinsics.......................................................................................... 335
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>__log2f</td>
<td>350</td>
</tr>
<tr>
<td>__logf</td>
<td>351</td>
</tr>
<tr>
<td>__powf</td>
<td>351</td>
</tr>
<tr>
<td>__saturatef</td>
<td>351</td>
</tr>
<tr>
<td>__sincosf</td>
<td>352</td>
</tr>
<tr>
<td>__sinf</td>
<td>352</td>
</tr>
<tr>
<td>__tanf</td>
<td>353</td>
</tr>
<tr>
<td>1.9. Double Precision Intrinsics</td>
<td>353</td>
</tr>
<tr>
<td>__dadd_rd</td>
<td>353</td>
</tr>
<tr>
<td>__dadd_rn</td>
<td>354</td>
</tr>
<tr>
<td>__dadd_ru</td>
<td>354</td>
</tr>
<tr>
<td>__dadd_rz</td>
<td>354</td>
</tr>
<tr>
<td>__ddiv_rd</td>
<td>355</td>
</tr>
<tr>
<td>__ddiv_rn</td>
<td>355</td>
</tr>
<tr>
<td>__ddiv_ru</td>
<td>356</td>
</tr>
<tr>
<td>__ddiv_rz</td>
<td>356</td>
</tr>
<tr>
<td>__dmul_rd</td>
<td>356</td>
</tr>
<tr>
<td>__dmul_rn</td>
<td>357</td>
</tr>
<tr>
<td>__dmul_ru</td>
<td>357</td>
</tr>
<tr>
<td>__dmul_rz</td>
<td>358</td>
</tr>
<tr>
<td>__drcp_rd</td>
<td>358</td>
</tr>
<tr>
<td>__drcp_rn</td>
<td>359</td>
</tr>
<tr>
<td>__drcp_ru</td>
<td>359</td>
</tr>
<tr>
<td>__drcp_rz</td>
<td>359</td>
</tr>
<tr>
<td>__dsqrt_rd</td>
<td>360</td>
</tr>
<tr>
<td>__dsqrt_rn</td>
<td>360</td>
</tr>
<tr>
<td>__dsqrt_ru</td>
<td>361</td>
</tr>
<tr>
<td>__dsqrt_rz</td>
<td>361</td>
</tr>
<tr>
<td>__dsub_rd</td>
<td>362</td>
</tr>
<tr>
<td>__dsub_rn</td>
<td>362</td>
</tr>
<tr>
<td>__dsub_ru</td>
<td>362</td>
</tr>
<tr>
<td>__dsub_rz</td>
<td>363</td>
</tr>
<tr>
<td>__fma_rd</td>
<td>363</td>
</tr>
<tr>
<td>__fma_rn</td>
<td>364</td>
</tr>
<tr>
<td>__fma_ru</td>
<td>364</td>
</tr>
<tr>
<td>__fma_rz</td>
<td>365</td>
</tr>
<tr>
<td>1.10. Integer Intrinsics</td>
<td>366</td>
</tr>
<tr>
<td>__brev</td>
<td>366</td>
</tr>
</tbody>
</table>
1.11. Type Casting Intrinsics
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>__double2float_rd</td>
<td>378</td>
</tr>
<tr>
<td>__double2float_rn</td>
<td>378</td>
</tr>
<tr>
<td>__double2float_ru</td>
<td>378</td>
</tr>
<tr>
<td>__double2float_rz</td>
<td>378</td>
</tr>
<tr>
<td>__double2hiint</td>
<td>379</td>
</tr>
<tr>
<td>__double2int_rd</td>
<td>379</td>
</tr>
<tr>
<td>__double2int_rn</td>
<td>379</td>
</tr>
<tr>
<td>__double2int_ru</td>
<td>380</td>
</tr>
<tr>
<td>__double2int_rz</td>
<td>380</td>
</tr>
<tr>
<td>__double2ll_rd</td>
<td>380</td>
</tr>
<tr>
<td>__double2ll_rn</td>
<td>380</td>
</tr>
<tr>
<td>__double2ll_ru</td>
<td>381</td>
</tr>
<tr>
<td>__double2ll_rz</td>
<td>381</td>
</tr>
<tr>
<td>__double2loint</td>
<td>381</td>
</tr>
<tr>
<td>__double2uint_rd</td>
<td>382</td>
</tr>
<tr>
<td>__double2uint_rn</td>
<td>382</td>
</tr>
<tr>
<td>__double2uint_ru</td>
<td>382</td>
</tr>
<tr>
<td>__double2uint_rz</td>
<td>382</td>
</tr>
<tr>
<td>__double2ull_rd</td>
<td>383</td>
</tr>
<tr>
<td>__double2ull_rn</td>
<td>383</td>
</tr>
<tr>
<td>__double2ull_ru</td>
<td>383</td>
</tr>
<tr>
<td>__double2ull_rz</td>
<td>384</td>
</tr>
<tr>
<td>__double_as_longlong</td>
<td>384</td>
</tr>
<tr>
<td>__float2int_rd</td>
<td>384</td>
</tr>
<tr>
<td>__float2int_rn</td>
<td>384</td>
</tr>
<tr>
<td>__float2int_ru</td>
<td>385</td>
</tr>
<tr>
<td>__float2int_rz</td>
<td>385</td>
</tr>
<tr>
<td>__float2ll_rd</td>
<td>385</td>
</tr>
<tr>
<td>__float2ll_rn</td>
<td>386</td>
</tr>
<tr>
<td>__float2ll_ru</td>
<td>386</td>
</tr>
<tr>
<td>__float2ll_rz</td>
<td>386</td>
</tr>
<tr>
<td>__float2uint_rd</td>
<td>386</td>
</tr>
<tr>
<td>__float2uint_rn</td>
<td>386</td>
</tr>
<tr>
<td>__float2uint_ru</td>
<td>387</td>
</tr>
<tr>
<td>__float2uint_rz</td>
<td>387</td>
</tr>
<tr>
<td>__float2ull_rd</td>
<td>388</td>
</tr>
<tr>
<td>__float2ull_rn</td>
<td>388</td>
</tr>
<tr>
<td>__float2ull_ru</td>
<td>388</td>
</tr>
</tbody>
</table>
1.12. SIMD Intrinsics

__vabs2
__vabs4
__vabsdiffs2
__vabsdiffs4
__vmins2.................................................................................................................................426
__vmins4.................................................................................................................................426
__vminu2.................................................................................................................................426
__vminu4.................................................................................................................................426
__vneg2......................................................................................................................................427
__vneg4......................................................................................................................................427
__vnegss2.................................................................................................................................. 427
__vnegss4.................................................................................................................................. 428
__vsads2.................................................................................................................................... 428
__vsads4.................................................................................................................................... 428
__vsadu2....................................................................................................................................429
__vsadu4....................................................................................................................................429
__vseteq2...................................................................................................................................429
__vseteq4...................................................................................................................................430
__vsetges2................................................................................................................................430
__vsetges4................................................................................................................................430
__vsetgeu2................................................................................................................................431
__vsetgeu4................................................................................................................................431
__vsetgtu2................................................................................................................................432
__vsetgtu4................................................................................................................................432
__vsetlts2..................................................................................................................................433
__vsetlts4..................................................................................................................................433
__vsetleu2..................................................................................................................................434
__vsetleu4..................................................................................................................................434
__vsetltls2................................................................................................................................434
__vsetltls4................................................................................................................................434
__vsetltlu2................................................................................................................................435
__vsetltlu4................................................................................................................................435
__vsetltu2.................................................................................................................................. 435
__vsetltu4.................................................................................................................................. 435
__vsetne2...................................................................................................................................435
__vsetne4...................................................................................................................................436
__vsub2......................................................................................................................................436
__vsub4......................................................................................................................................436
__vsubss2..................................................................................................................................437
__vsubss4..................................................................................................................................437
__vsubus2..................................................................................................................................437
__vsubus4..................................................................................................................................438
<table>
<thead>
<tr>
<th>Operator</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>operator=</td>
<td>446</td>
</tr>
<tr>
<td>__half2</td>
<td>447</td>
</tr>
<tr>
<td>x</td>
<td>447</td>
</tr>
<tr>
<td>y</td>
<td>447</td>
</tr>
<tr>
<td>__half2</td>
<td>447</td>
</tr>
<tr>
<td>__half2</td>
<td>447</td>
</tr>
<tr>
<td>__half2</td>
<td>447</td>
</tr>
<tr>
<td>__half2</td>
<td>447</td>
</tr>
<tr>
<td>__half2</td>
<td>447</td>
</tr>
<tr>
<td>operator __half2_raw</td>
<td>448</td>
</tr>
<tr>
<td>operator=</td>
<td>448</td>
</tr>
<tr>
<td>__half2_raw</td>
<td>448</td>
</tr>
<tr>
<td>__half_raw</td>
<td>448</td>
</tr>
<tr>
<td>__nv_bfloat16</td>
<td>449</td>
</tr>
<tr>
<td>__x</td>
<td>449</td>
</tr>
<tr>
<td>__nv_bfloat16</td>
<td>449</td>
</tr>
<tr>
<td>__nv_bfloat16</td>
<td>449</td>
</tr>
<tr>
<td>__nv_bfloat16</td>
<td>449</td>
</tr>
<tr>
<td>__nv_bfloat16</td>
<td>449</td>
</tr>
<tr>
<td>__nv_bfloat16</td>
<td>450</td>
</tr>
<tr>
<td>__nv_bfloat16</td>
<td>450</td>
</tr>
<tr>
<td>__nv_bfloat16</td>
<td>450</td>
</tr>
<tr>
<td>__nv_bfloat16</td>
<td>450</td>
</tr>
<tr>
<td>__nv_bfloat16</td>
<td>450</td>
</tr>
<tr>
<td>__nv_bfloat16</td>
<td>451</td>
</tr>
<tr>
<td>operator __nv_bfloat16_raw</td>
<td>451</td>
</tr>
<tr>
<td>operator __nv_bfloat16_raw</td>
<td>451</td>
</tr>
<tr>
<td>operator bool</td>
<td>451</td>
</tr>
<tr>
<td>operator char</td>
<td>451</td>
</tr>
<tr>
<td>operator float</td>
<td>452</td>
</tr>
<tr>
<td>operator int</td>
<td>452</td>
</tr>
<tr>
<td>operator long</td>
<td>452</td>
</tr>
<tr>
<td>operator long long</td>
<td>452</td>
</tr>
<tr>
<td>operator short</td>
<td>452</td>
</tr>
</tbody>
</table>
operator signed char ................................................................. 452
operator unsigned char ............................................................ 453
operator unsigned int ............................................................... 453
operator unsigned long ............................................................ 453
operator unsigned long long .................................................... 453
operator unsigned short .......................................................... 453
operator= .................................................................................. 454
operator= .................................................................................. 454
operator= .................................................................................. 454
operator= .................................................................................. 454
operator= .................................................................................. 454
operator= .................................................................................. 454
operator= .................................................................................. 454
operator= .................................................................................. 454
operator= .................................................................................. 454
operator= .................................................................................. 454
__nv_bfloat162 ......................................................................... 456
x ............................................................................................... 456
y ............................................................................................... 456
__nv_bfloat162 ......................................................................... 456
__nv_bfloat162 ......................................................................... 456
__nv_bfloat162 ......................................................................... 456
__nv_bfloat162 ......................................................................... 456
__nv_bfloat162 ......................................................................... 456
__nv_bfloat162 ......................................................................... 456
operator __nv_bfloat162_raw .................................................. 457
operator= .................................................................................. 457
operator= .................................................................................. 457
__nv_bfloat162 raw .................................................................. 457
__nv_bfloat16 raw ...................................................................... 457
__nv_fp8_e4m3 .......................................................................... 458
__x ............................................................................................. 458
__nv_fp8_e4m3 .......................................................................... 458
__nv_fp8_e4m3 .......................................................................... 458
__nv_fp8_e4m3 .......................................................................... 458
__nv_fp8_e4m3 .......................................................................... 458
__nv_fp8_e4m3 .......................................................................... 458
__nv_fp8_e4m3 .......................................................................... 458
__nv_fp8_e4m3 .......................................................................... 458
__nv_fp8_e4m3 .......................................................................... 459
__nv_fp8_e4m3 .......................................................................... 459
__nv_fp8_e4m3 .......................................................................... 459
__nv_fp8_e4m3......................................................................................................................... 459
operator __half......................................................................................................................... 460
operator __nv_bfloat16.............................................................................................................460
operator bool.............................................................................................................................461
operator char............................................................................................................................ 461
operator double........................................................................................................................ 461
operator float............................................................................................................................ 461
operator int............................................................................................................................... 461
operator long int.........................................................................................................................461
operator long long int..............................................................................................................462
operator short int.......................................................................................................................462
operator signed char.................................................................................................................462
operator unsigned char.............................................................................................................462
operator unsigned int.................................................................................................................462
operator unsigned long int.......................................................................................................462
operator unsigned long long int...............................................................................................463
operator unsigned short int......................................................................................................463
__nv_fp8_e5m2............................................................................................................................. 463
__x..............................................................................................................................................463
__nv_fp8_e5m2............................................................................................................................. 463
__nv_fp8_e5m2............................................................................................................................. 463
__nv_fp8_e5m2............................................................................................................................. 464
__nv_fp8_e5m2............................................................................................................................. 464
__nv_fp8_e5m2............................................................................................................................. 464
__nv_fp8_e5m2............................................................................................................................. 464
__nv_fp8_e5m2............................................................................................................................. 464
__nv_fp8_e5m2............................................................................................................................. 465
__nv_fp8_e5m2............................................................................................................................. 465
__nv_fp8_e5m2............................................................................................................................. 465
__nv_fp8_e5m2............................................................................................................................. 465
__nv_fp8_e5m2............................................................................................................................. 465
__nv_fp8_e5m2............................................................................................................................. 466
operator __half......................................................................................................................... 466
operator __nv_bfloat16.............................................................................................................466
operator bool.............................................................................................................................466
operator char............................................................................................................................ 466
operator double........................................................................................................................ 466
operator float............................................................................................................................ 466
operator int............................................................................................................................... 467
operator long int......................................................................................................................... 467
operator long long int.................................................................................................................. 467
operator short int......................................................................................................................... 467
operator signed char.................................................................................................................... 467
operator unsigned char................................................................................................................. 467
operator unsigned int..................................................................................................................... 468
operator unsigned long int............................................................................................................ 468
operator unsigned long long int..................................................................................................... 468
operator unsigned short int............................................................................................................ 468
__nv_fp8x2_e4m3..........................................................................................................................468
__x..............................................................................................................................................468
__nv_fp8x2_e4m3......................................................................................................................469
__nv_fp8x2_e4m3......................................................................................................................469
__nv_fp8x2_e4m3......................................................................................................................469
__nv_fp8x2_e4m3......................................................................................................................469
__nv_fp8x2_e4m3......................................................................................................................469
__nv_fp8x2_e4m3......................................................................................................................469
__nv_fp8x2_e5m2..........................................................................................................................470
__x..............................................................................................................................................470
__nv_fp8x2_e5m2......................................................................................................................470
__nv_fp8x2_e5m2......................................................................................................................470
__nv_fp8x2_e5m2......................................................................................................................470
__nv_fp8x2_e5m2......................................................................................................................470
__nv_fp8x2_e5m2......................................................................................................................470
__nv_fp8x2_e5m2......................................................................................................................470
__nv_fp8x2_e5m2......................................................................................................................471
__nv_fp8x4_e4m3..........................................................................................................................471
__x..............................................................................................................................................471
__nv_fp8x4_e4m3......................................................................................................................471
__nv_fp8x4_e4m3......................................................................................................................471
__nv_fp8x4_e4m3......................................................................................................................472
__nv_fp8x4_e4m3......................................................................................................................472
Chapter 3. Data Fields

 CUDA Math API

v12.4 | xl
Chapter 1. Modules

Here is a list of all modules:

- **FP8 Intrinsic**
  - FP8 Conversion and Data Movement
  - C++ struct for handling fp8 data type of e5m2 kind.
  - C++ struct for handling vector type of two fp8 values of e5m2 kind.
  - C++ struct for handling vector type of four fp8 values of e5m2 kind.
  - C++ struct for handling fp8 data type of e4m3 kind.
  - C++ struct for handling vector type of two fp8 values of e4m3 kind.
  - C++ struct for handling vector type of four fp8 values of e4m3 kind.

- **Half Precision Intrinsic**
  - Half Arithmetic Constants
  - Half Arithmetic Functions
  - Half2 Arithmetic Functions
  - Half Comparison Functions
  - Half2 Comparison Functions
  - Half Precision Conversion and Data Movement
  - Half Math Functions
  - Half2 Math Functions

- **Bfloat16 Precision Intrinsic**
  - Bfloat16 Arithmetic Constants
  - Bfloat16 Arithmetic Functions
  - Bfloat162 Arithmetic Functions
  - Bfloat16 Comparison Functions
1.1. FP8 Intrinsics

This section describes fp8 intrinsic functions. To use these functions, include the header file cuda_fp8.h in your program. The following macros are available to help users selectively enable/disable various definitions present in the header file:

- __CUDA_NO_FP8_CONVERSIONS__ - If defined, this macro will prevent any use of the C++ type conversions (converting constructors and conversion operators) defined in the header.
- __CUDA_NO_FP8_CONVERSION_OPERATORS__ - If defined, this macro will prevent any use of the C++ conversion operators from fp8 to other types.

FP8 Conversion and Data Movement

C++ struct for handling fp8 data type of e5m2 kind.

C++ struct for handling vector type of two fp8 values of e5m2 kind.
C++ struct for handling vector type of four fp8 values of e5m2 kind.

C++ struct for handling fp8 data type of e4m3 kind.

C++ struct for handling vector type of two fp8 values of e4m3 kind.

C++ struct for handling vector type of four fp8 values of e4m3 kind.

1.1.1. FP8 Conversion and Data Movement

FP8 Intrinsics

To use these functions, include the header file `cuda_fp8.h` in your program.

`enum __nv_fp8_interpretation_t`

Enumerates the possible interpretations of the 8-bit values when referring to them as fp8 types.

Values

`__NV_E4M3`
Stands for fp8 numbers of e4m3 kind.

`__NV_E5M2`
Stands for fp8 numbers of e5m2 kind.

`enum __nv_saturation_t`

Enumerates the modes applicable when performing a narrowing conversion to fp8 destination types.

Values

`__NV_NOSAT`
Means no saturation to finite is performed when conversion results in rounding values outside the range of destination type. NOTE: for fp8 type of e4m3 kind, the results that are larger than the maximum representable finite number of the target format become NaN.

`__NV_SATFINITE`
Means input larger than the maximum representable finite number MAXNORM of the target format round to the MAXNORM of the same sign as input.

typedef unsigned char __nv_fp8_storage_t
8-bit unsigned integer type abstraction used to for fp8 floating-point numbers storage.

typedef unsigned short int __nv_fp8x2_storage_t
16-bit unsigned integer type abstraction used to for storage of pairs of fp8 floating-point numbers.

typedef unsigned int __nv_fp8x4_storage_t
32-bit unsigned integer type abstraction used to for storage of tetrads of fp8 floating-point numbers.

__host____device____nv_fp8x2_storage_t
__nv_cvt_bfloat16raw2_to_fp8x2 (const __nv_bfloat162_raw x, const __nv_saturation_t saturate, const __nv_fp8_interpretation_t fp8_interpretation)
Converts input vector of two nv_bfloat16 precision numbers packed in __nv_bfloat162_raw x into a vector of two values of fp8 type of the requested kind using round-to-nearest-even rounding and requested saturation mode.

Returns

- The __nv_fp8x2_storage_t value holds the result of conversion.

Description

Converts input vector x to a vector of two fp8 values of the kind specified by fp8_interpretation parameter, using round-to-nearest-even rounding and saturation mode specified by saturate parameter.

__host____device____nv_fp8_storage_t
__nv_cvt_bfloat16raw_to_fp8 (const __nv_bfloat16_raw x, const __nv_saturation_t saturate, const __nv_fp8_interpretation_t fp8_interpretation)
Converts input nv_bfloat16 precision x to fp8 type of the requested kind using round-to-nearest-even rounding and requested saturation mode.

Returns

- The __nv_fp8_storage_t value holds the result of conversion.
Description
Converts input `x` to fp8 type of the kind specified by `fp8_interpretation` parameter, using round-to-nearest-even rounding and saturation mode specified by `saturate` parameter.

```c
__host__device__nv_fp8x2_storage_t
__nv_cvt_double2_to_fp8x2 (const double2 x, const __nv_saturation_t saturate, const __nv_fp8_interpretation_t fp8_interpretation)
```
Converts input vector of two `double` precision numbers packed in `double2 x` into a vector of two values of fp8 type of the requested kind using round-to-nearest-even rounding and requested saturation mode.

Returns
- The `__nv_fp8x2_storage_t` value holds the result of conversion.

Description
Converts input vector `x` to a vector of two fp8 values of the kind specified by `fp8_interpretation` parameter, using round-to-nearest-even rounding and saturation mode specified by `saturate` parameter.

```c
__host__device__nv_fp8_storage_t
__nv_cvt_double_to_fp8 (const double x, const __nv_saturation_t saturate, const __nv_fp8_interpretation_t fp8_interpretation)
```
Converts input `double` precision `x` to fp8 type of the requested kind using round-to-nearest-even rounding and requested saturation mode.

Returns
- The `__nv_fp8_storage_t` value holds the result of conversion.

Description
Converts input `x` to fp8 type of the kind specified by `fp8_interpretation` parameter, using round-to-nearest-even rounding and saturation mode specified by `saturate` parameter.
__host__ __device__ __nv_fp8x2_storage_t __nv_cvt_float2_to_fp8x2 (const float2 x, const __nv_saturation_t saturate, const __nv_fp8_interpretation_t fp8_interpretation)

Converts input vector of two single precision numbers packed in float2 x into a vector of two values of fp8 type of the requested kind using round-to-nearest-even rounding and requested saturation mode.

**Returns**

- The __nv_fp8x2_storage_t value holds the result of conversion.

**Description**

Converts input vector x to a vector of two fp8 values of the kind specified by fp8_interpretation parameter, using round-to-nearest-even rounding and saturation mode specified by saturate parameter.

__host__ __device__ __nv_fp8_storage_t __nv_cvt_float_to_fp8 (const float x, const __nv_saturation_t saturate, const __nv_fp8_interpretation_t fp8_interpretation)

Converts input single precision x to fp8 type of the requested kind using round-to-nearest-even rounding and requested saturation mode.

**Returns**

- The __nv_fp8_storage_t value holds the result of conversion.

**Description**

Converts input x to fp8 type of the kind specified by fp8_interpretation parameter, using round-to-nearest-even rounding and saturation mode specified by saturate parameter.
__host__ __device__ nv_cvt_fp8_to_halfraw (const __nv_fp8_storage_t x, const __nv_fp8_interpretation_t fp8_interpretation)
Converts input fp8 x of the specified kind to half precision.

Returns

- The __half_raw value holds the result of conversion.

Description

Converts input x of fp8 type of the kind specified by fp8_interpretation parameter to half precision.

__host__ __device__ nv_cvt_fp8x2_to_halfraw2 (const __nv_fp8x2_storage_t x, const __nv_fp8_interpretation_t fp8_interpretation)
Converts input vector of two fp8 values of the specified kind to a vector of two half precision values packed in __half2_raw structure.

Returns

- The __half2_raw value holds the result of conversion.

Description

Converts input vector x of fp8 type of the kind specified by fp8_interpretation parameter to a vector of two half precision values and returns as __half2_raw structure.

__host__ __device__ __nv_fp8x2_storage_t __nv_cvt_halfraw2_to_fp8x2 (const __half2_raw x, const __nv_saturation_t saturate, const __nv_fp8_interpretation_t fp8_interpretation)
Converts input vector of two half precision numbers packed in __half2_raw x into a vector of two values of fp8 type of the requested kind using round-to-nearest-even rounding and requested saturation mode.

Returns

- The __nv_fp8x2_storage_t value holds the result of conversion.
Description

Converts input vector $x$ to a vector of two fp8 values of the kind specified by fp8_interpretation parameter, using round-to-nearest-even rounding and saturation mode specified by saturate parameter.

```c
__host__ __device__ __nv_fp8_storage_t
__nv_cvt_halfraw_to_fp8 (const __half_raw x, const __nv_saturation_t saturate, const __nv_fp8_interpretation_t fp8_interpretation)
```

Converts input half precision $x$ to fp8 type of the requested kind using round-to-nearest-even rounding and requested saturation mode.

Returns

- The __nv_fp8_storage_t value holds the result of conversion.

Description

Converts input $x$ to fp8 type of the kind specified by fp8_interpretation parameter, using round-to-nearest-even rounding and saturation mode specified by saturate parameter.

1.1.2. C++ struct for handling fp8 data type of e5m2 kind.

FP8 Intrinsics

```c
struct __nv_fp8_e5m2
__nv_fp8_e5m2 datatype
```

```c
__nv_fp8_storage_t :: __nv_fp8_e5m2 :: __x
```

Storage variable contains the fp8 floating-point data.

```c
__host__ __device__ __nv_fp8_e5m2::__nv_fp8_e5m2 (const long long int val)
```

Description

Constructor from long long int data type, relies on __NV_SATFINITE behavior for out-of-range values.
__host__ __device__ nv_fp8_e5m2::__nv_fp8_e5m2 (const long int val)

Description
Constructor from long int data type, relies on __NV_SATFINITE behavior for out-of-range values.

__host__ __device__ nv_fp8_e5m2::__nv_fp8_e5m2 (const int val)

Description
Constructor from int data type, relies on __NV_SATFINITE behavior for out-of-range values.

__host__ __device__ nv_fp8_e5m2::__nv_fp8_e5m2 (const short int val)

Description
Constructor from short int data type.

__host__ __device__ nv_fp8_e5m2::__nv_fp8_e5m2 (const unsigned long long int val)

Description
Constructor from unsigned long long int data type, relies on __NV_SATFINITE behavior for out-of-range values.

__host__ __device__ nv_fp8_e5m2::__nv_fp8_e5m2 (const unsigned long int val)

Description
Constructor from unsigned long int data type, relies on __NV_SATFINITE behavior for out-of-range values.
__host__ __device__ __nv_fp8_e5m2::__nv_fp8_e5m2 (const unsigned int val)

Description
Constructor from unsigned int data type, relies on __NV_SATFINITE behavior for out-of-range values.

__host__ __device__ __nv_fp8_e5m2::__nv_fp8_e5m2 (const unsigned short int val)

Description
Constructor from unsigned short int data type, relies on __NV_SATFINITE behavior for out-of-range values.

__host__ __device__ __nv_fp8_e5m2::__nv_fp8_e5m2 (const double f)

Description
Constructor from double data type, relies on __NV_SATFINITE behavior for out-of-range values.

__host__ __device__ __nv_fp8_e5m2::__nv_fp8_e5m2 (const float f)

Description
Constructor from float data type, relies on __NV_SATFINITE behavior for out-of-range values.

__host__ __device__ __nv_fp8_e5m2::__nv_fp8_e5m2 (const __nv_bfloat16 f)

Description
Constructor from __nv_bfloat16 data type, relies on __NV_SATFINITE behavior for out-of-range values.
__host__ __device__ __nv_fp8_e5m2::__nv_fp8_e5m2 (const __half f)

**Description**
Constructor from __half data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8_e5m2::__nv_fp8_e5m2 ()

**Description**
Constructor by default.

__host__ __device__ __nv_fp8_e5m2::operator __half ()

**Description**
Conversion operator to __half data type.

__host__ __device__ __nv_fp8_e5m2::operator __nv_bfloat16 ()

**Description**
Conversion operator to __nv_bfloat16 data type.

__host__ __device__ __nv_fp8_e5m2::operator bool ()

**Description**
Conversion operator to bool data type. +0 and -0 inputs convert to false. Non-zero inputs convert to true.

__host__ __device__ __nv_fp8_e5m2::operator char ()

**Description**
Conversion operator to an implementation defined char data type. Detects signedness of the char type and proceeds accordingly, see further details in signed and unsigned char operators. Clamps inputs to the output range. NaN inputs convert to zero.
\texttt{\_\_host\_\_\_device\_\_\_nv\_fp8\_e5m2::operator\ double\ ()}

\textbf{Description}
Conversion operator to \texttt{double} data type.

\texttt{\_\_host\_\_\_device\_\_\_nv\_fp8\_e5m2::operator\ float\ ()}

\textbf{Description}
Conversion operator to \texttt{float} data type.

\texttt{\_\_host\_\_\_device\_\_\_nv\_fp8\_e5m2::operator\ int\ ()}

\textbf{Description}
Conversion operator to \texttt{int} data type. Clamps too large inputs to the output range. \texttt{NaN} inputs convert to zero.

\texttt{\_\_host\_\_\_device\_\_\_nv\_fp8\_e5m2::operator\ long\ int\ ()}

\textbf{Description}
Conversion operator to \texttt{long\ int} data type. Clamps too large inputs to the output range. \texttt{NaN} inputs convert to zero if output type is 32-bit. \texttt{NaN} inputs convert to \texttt{0x8000000000000000ULL} if output type is 64-bit.

\texttt{\_\_host\_\_\_device\_\_\_nv\_fp8\_e5m2::operator\ long\ long\ int\ ()}

\textbf{Description}
Conversion operator to \texttt{long\ long\ int} data type. Clamps too large inputs to the output range. \texttt{NaN} inputs convert to \texttt{0x8000000000000000LL}.

\texttt{\_\_host\_\_\_device\_\_\_nv\_fp8\_e5m2::operator\ short\ int\ ()}

\textbf{Description}
Conversion operator to \texttt{short\ int} data type. Clamps too large inputs to the output range. \texttt{NaN} inputs convert to zero.
__host____device____nv_fp8_e5m2::operator signed char ()

Description
Conversion operator to signed char data type. Clamps too large inputs to the output range. NaN inputs convert to zero.

__host____device____nv_fp8_e5m2::operator unsigned char ()

Description
Conversion operator to unsigned char data type. Clamps negative and too large inputs to the output range. NaN inputs convert to zero.

__host____device____nv_fp8_e5m2::operator unsigned int ()

Description
Conversion operator to unsigned int data type. Clamps negative and too large inputs to the output range. NaN inputs convert to zero.

__host____device____nv_fp8_e5m2::operator unsigned long int ()

Description
Conversion operator to unsigned long int data type. Clamps negative and too large inputs to the output range. NaN inputs convert to zero if output type is 32-bit. NaN inputs convert to 0x8000000000000000ULL if output type is 64-bit.

__host____device____nv_fp8_e5m2::operator unsigned long long int ()

Description
Conversion operator to unsigned long long int data type. Clamps negative and too large inputs to the output range. NaN inputs convert to 0x8000000000000000ULL.
__host__ __device__ nv_fp8_e5m2::operator unsigned short int ()

Description
Conversion operator to unsigned short int data type. Clamps negative and too large inputs to the output range. NaN inputs convert to zero.

1.1.3. C++ struct for handling vector type of two fp8 values of e5m2 kind.

FP8 Intrinsics

struct __nv_fp8x2_e5m2
__nv_fp8x2_e5m2 datatype

__nv_fp8x2_storage_t :: __nv_fp8x2_e5m2::__x
Storage variable contains the vector of two fp8 floating-point data values.

__host__ __device__ __nv_fp8x2_e5m2::__nv_fp8x2_e5m2 (const double2 f)

Description
Constructor from double2 data type, relies on __NV_SATFINITE behavior for out-of-range values.

__host__ __device__ __nv_fp8x2_e5m2::__nv_fp8x2_e5m2 (const float2 f)

Description
Constructor from float2 data type, relies on __NV_SATFINITE behavior for out-of-range values.
__host__ __device__ nv_fp8x2_e5m2::nv_fp8x2_e5m2 (const __nv_bfloat16 f)

Description
Constructor from __nv_bfloat16 data type, relies on __NV_SATFINITE behavior for out-of-range values.

__host__ __device__ nv_fp8x2_e5m2::nv_fp8x2_e5m2 (const __half2 f)

Description
Constructor from __half2 data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8x2_e5m2::nv_fp8x2_e5m2 ()

Description
Constructor by default.

__host__ __device__ nv_fp8x2_e5m2::operator __half2 ()

Description
Conversion operator to __half2 data type.

__host__ __device__ nv_fp8x2_e5m2::operator float2 ()

Description
Conversion operator to float2 data type.

1.1.4. C++ struct for handling vector type of four fp8 values of e5m2 kind.

FP8 Intrinsics
struct __nv_fp8x4_e5m2
__nv_fp8x4_e5m2 datatype

__nv_fp8x4_storage_t :: __nv_fp8x4_e5m2::__x
Storage variable contains the vector of four fp8 floating-point data values.

__host__ __device__ __nv_fp8x4_e5m2::__nv_fp8x4_e5m2 (const double4 f)

Description
Constructor from double4 vector data type, relies on __NV_SATFINITE behavior for out-of-range values.

__host__ __device__ __nv_fp8x4_e5m2::__nv_fp8x4_e5m2 (const float4 f)

Description
Constructor from float4 vector data type, relies on __NV_SATFINITE behavior for out-of-range values.

__host__ __device__ __nv_fp8x4_e5m2::__nv_fp8x4_e5m2 (const __nv_bfloat162 flo, const __nv_bfloat162 fhi)

Description
Constructor from a pair of __nv_bfloat162 data type values, relies on __NV_SATFINITE behavior for out-of-range values.

__host__ __device__ __nv_fp8x4_e5m2::__nv_fp8x4_e5m2 (const __half2 flo, const __half2 fhi)

Description
Constructor from a pair of __half2 data type values, relies on __NV_SATFINITE behavior for out-of-range values.
__nv_fp8x4_e5m2::__nv_fp8x4_e5m2 ()

Description
Constructor by default.

__host__ __device__ nv_fp8x4_e5m2::operator float4 ()

Description
Conversion operator to float4 vector data type.

1.1.5. C++ struct for handling fp8 data type of e4m3 kind.

FP8 Intrinsics

struct __nv_fp8_e4m3
__nv_fp8_e4m3 datatype

__nv_fp8_storage_t ::__nv_fp8_e4m3::__x

Storage variable contains the fp8 floating-point data.

__host__ __device__ nv_fp8_e4m3::__nv_fp8_e4m3::__nv_fp8_e4m3 (const long long int val)

Description
Constructor from long long int data type, relies on __NV_SATFINITE behavior for out-of-range values.

__host__ __device__ nv_fp8_e4m3::__nv_fp8_e4m3::__nv_fp8_e4m3 (const long int val)

Description
Constructor from long int data type, relies on __NV_SATFINITE behavior for out-of-range values.
__host__ __device__ nv_fp8_e4m3::nv_fp8_e4m3 (const int val)

Description
Constructor from int data type, relies on __NV_SATFINITE behavior for out-of-range values.

__host__ __device__ nv_fp8_e4m3::nv_fp8_e4m3 (const short int val)

Description
Constructor from short int data type, relies on __NV_SATFINITE behavior for out-of-range values.

__host__ __device__ nv_fp8_e4m3::nv_fp8_e4m3 (const unsigned long long int val)

Description
Constructor from unsigned long long int data type, relies on __NV_SATFINITE behavior for out-of-range values.

__host__ __device__ nv_fp8_e4m3::nv_fp8_e4m3 (const unsigned long int val)

Description
Constructor from unsigned long int data type, relies on __NV_SATFINITE behavior for out-of-range values.

__host__ __device__ nv_fp8_e4m3::nv_fp8_e4m3 (const unsigned int val)

Description
Constructor from unsigned int data type, relies on __NV_SATFINITE behavior for out-of-range values.
__host__ __device__ __nv_fp8_e4m3::__nv_fp8_e4m3 (const unsigned short int val)

Description
Constructor from unsigned short int data type, relies on __NV_SATFINITE behavior for out-of-range values.

__host__ __device__ __nv_fp8_e4m3::__nv_fp8_e4m3 (const double f)

Description
Constructor from double data type, relies on __NV_SATFINITE behavior for out-of-range values.

__host__ __device__ __nv_fp8_e4m3::__nv_fp8_e4m3 (const float f)

Description
Constructor from float data type, relies on __NV_SATFINITE behavior for out-of-range values.

__host__ __device__ __nv_fp8_e4m3::__nv_fp8_e4m3 (const __nv_bfloat16 f)

Description
Constructor from __nv_bfloat16 data type, relies on __NV_SATFINITE behavior for out-of-range values.

__host__ __device__ __nv_fp8_e4m3::__nv_fp8_e4m3 (const __half f)

Description
Constructor from __half data type, relies on __NV_SATFINITE behavior for out-of-range values.
__nv_fp8_e4m3::__nv_fp8_e4m3 ()

Description
Constructor by default.

__host____device____nv_fp8_e4m3::operator __half ()

Description
Conversion operator to __half data type.

__host____device____nv_fp8_e4m3::operator __nv_bfloat16 ()

Description
Conversion operator to __nv_bfloat16 data type.

__host____device____nv_fp8_e4m3::operator bool ()

Description
Conversion operator to bool data type. +0 and -0 inputs convert to false. Non-zero inputs convert to true.

__host____device____nv_fp8_e4m3::operator char ()

Description
Conversion operator to an implementation defined char data type.
Detects signedness of the char type and proceeds accordingly, see further details in signed and unsigned char operators.
Clamps inputs to the output range. NaN inputs convert to zero.

__host____device____nv_fp8_e4m3::operator double ()

Description
Conversion operator to double data type.
__host__ device nv_fp8_e4m3::operator float ()

Description
Conversion operator to float data type.

__host__ device nv_fp8_e4m3::operator int ()

Description
Conversion operator to int data type. NaN inputs convert to zero.

__host__ device nv_fp8_e4m3::operator long int ()

Description
Conversion operator to long int data type. Clamps too large inputs to the output range. NaN inputs convert to zero if output type is 32-bit. NaN inputs convert to 0x8000000000000000ULL if output type is 64-bit.

__host__ device nv_fp8_e4m3::operator long long int ()

Description
Conversion operator to long long int data type. NaN inputs convert to 0x8000000000000000LL.

__host__ device nv_fp8_e4m3::operator short int ()

Description
Conversion operator to short int data type. NaN inputs convert to zero.

__host__ device nv_fp8_e4m3::operator signed char ()

Description
Conversion operator to signed char data type. Clamps too large inputs to the output range. NaN inputs convert to zero.
__host__ __device__ nv_fp8_e4m3::operator unsigned char ()

Description
Conversion operator to unsigned char data type. Clamps negative and too large inputs to the output range. NaN inputs convert to zero.

__host__ __device__ nv_fp8_e4m3::operator unsigned int ()

Description
Conversion operator to unsigned int data type. Clamps negative inputs to zero. NaN inputs convert to zero.

__host__ __device__ nv_fp8_e4m3::operator unsigned long int ()

Description
Conversion operator to unsigned long int data type. Clamps negative and too large inputs to the output range. NaN inputs convert to zero if output type is 32-bit. NaN inputs convert to 0x8000000000000000ULL if output type is 64-bit.

__host__ __device__ nv_fp8_e4m3::operator unsigned long long int ()

Description
Conversion operator to unsigned long long int data type. Clamps negative inputs to zero. NaN inputs convert to 0x8000000000000000ULL.

__host__ __device__ nv_fp8_e4m3::operator unsigned short int ()

Description
Conversion operator to unsigned short int data type. Clamps negative inputs to zero. NaN inputs convert to zero.
1.1.6. C++ struct for handling vector type of two fp8 values of e4m3 kind.

FP8 Intrinsics

struct __nv_fp8x2_e4m3
datatype

__nv_fp8x2_storage_t::__nv_fp8x2_e4m3::__x

Storage variable contains the vector of two fp8 floating-point data values.

__host____device____nv_fp8x2_e4m3::__nv_fp8x2_e4m3(const double2 f)

Description
Constructor from double2 data type, relies on __NV_SATFINITE behavior for out-of-range values.

__host____device____nv_fp8x2_e4m3::__nv_fp8x2_e4m3(const float2 f)

Description
Constructor from float2 data type, relies on __NV_SATFINITE behavior for out-of-range values.

__host____device____nv_fp8x2_e4m3::__nv_fp8x2_e4m3(const __nv_bfloat162 f)

Description
Constructor from __nv_bfloat162 data type, relies on __NV_SATFINITE behavior for out-of-range values.
__host__ __device__ __nv_fp8x2_e4m3::__nv_fp8x2_e4m3 (const __half2 f)

Description
Constructor from __half2 data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8x2_e4m3::__nv_fp8x2_e4m3 ()

Description
Constructor by default.

__host__ __device__ __nv_fp8x2_e4m3::operator __half2 ()

Description
Conversion operator to __half2 data type.

__host__ __device__ __nv_fp8x2_e4m3::operator float2 ()

Description
Conversion operator to float2 data type.

1.1.7. C++ struct for handling vector type of four fp8 values of e4m3 kind.

FP8 Intrinsics

struct __nv_fp8x4_e4m3
__nv_fp8x4_e4m3 datatype

__nv_fp8x4_storage_t ::__nv_fp8x4_e4m3::__x
Storage variable contains the vector of four fp8 floating-point data values.
__host__ __device__ __nv_fp8x4_e4m3::__nv_fp8x4_e4m3 (const double4 f)

Description
Constructor from double4 vector data type, relies on __NV_SATFINITE behavior for out-of-range values.

__host__ __device__ __nv_fp8x4_e4m3::__nv_fp8x4_e4m3 (const float4 f)

Description
Constructor from float4 vector data type, relies on __NV_SATFINITE behavior for out-of-range values.

__host__ __device__ __nv_fp8x4_e4m3::__nv_fp8x4_e4m3 (const __nv_bfloat16 flo, const __nv_bfloat16 fhi)

Description
Constructor from a pair of __nv_bfloat16 data type values, relies on __NV_SATFINITE behavior for out-of-range values.

__host__ __device__ __nv_fp8x4_e4m3::__nv_fp8x4_e4m3 (const __half2 flo, const __half2 fhi)

Description
Constructor from a pair of __half2 data type values, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8x4_e4m3::__nv_fp8x4_e4m3 ()

Description
Constructor by default.

__host__ __device__ __nv_fp8x4_e4m3::operator float4 ()

Description
Conversion operator to float4 vector data type.
1.2. Half Precision Intrinsics

This section describes half precision intrinsic functions. To use these functions, include the header file `cuda_fp16.h` in your program. All of the functions defined here are available in device code. Some of the functions are also available to host compilers, please refer to respective functions’ documentation for details.

NOTE: Aggressive floating-point optimizations performed by host or device compilers may affect numeric behavior of the functions implemented in this header.

The following macros are available to help users selectively enable/disable various definitions present in the header file:

- `CUDA_NO_HALF` - If defined, this macro will prevent the definition of additional type aliases in the global namespace, helping to avoid potential conflicts with symbols defined in the user program.
- `__CUDA_NO_HALF_CONVERSIONS__` - If defined, this macro will prevent the use of the C++ type conversions (converting constructors and conversion operators) that are common for built-in floating-point types, but may be undesirable for `half` which is essentially a user-defined type.
- `__CUDA_NO_HALF_OPERATORS__` and `__CUDA_NO_HALF2_OPERATORS__` - If defined, these macros will prevent the inadvertent use of usual arithmetic and comparison operators. This enforces the storage-only type semantics and prevents C++ style computations on `half` and `half2` types.

```c
struct __half
__half data type

struct __half2
__half2 data type

struct __half2_raw
__half2_raw data type

struct __half_raw
__half_raw data type
```

### Half Arithmetic Constants
Half Arithmetic Functions
Half2 Arithmetic Functions
Half Comparison Functions
Half2 Comparison Functions
Half Precision Conversion and Data Movement
Half Math Functions
Half2 Math Functions

typedef __nv_half
This datatype is an __nv__ prefixed alias.

typedef __nv_half2
This datatype is an __nv__ prefixed alias.

typedef __nv_half2_raw
This datatype is an __nv__ prefixed alias.

typedef __nv_half_raw
This datatype is an __nv__ prefixed alias.

typedef half
This datatype is meant to be the first-class or fundamental implementation of the half-precision numbers format.
Should be implemented in the compiler in the future. Current implementation is a simple typedef to a respective user-level type with underscores.
typedef half2
This datatype is meant to be the first-class or fundamental implementation of type for pairs of half-precision numbers.

Should be implemented in the compiler in the future. Current implementation is a simple typedef to a respective user-level type with underscores.

typedef nv_half
This datatype is an nv prefixed alias.

typedef nv_half2
This datatype is an nv prefixed alias.

1.2.1. Half Arithmetic Constants
Half Precision Intrinsics
To use these constants, include the header file cuda_fp16.h in your program.

#define CUDART_INF_FP16 __ushort_as_half((unsigned short)0x7C00U)
Defines floating-point positive infinity value for the half data type.

#define CUDART_MAX_NORMAL_FP16 __ushort_as_half((unsigned short)0x7BFFU)
Defines a maximum representable value for the half data type.

#define CUDART_MIN_DENORM_FP16 __ushort_as_half((unsigned short)0x0001U)
Defines a minimum representable (denormalized) value for the half data type.

#define CUDART_NAN_FP16 __ushort_as_half((unsigned short)0x7FFFU)
Defines canonical NaN value for the half data type.

#define CUDART_NEG_ZERO_FP16 __ushort_as_half((unsigned short)0x8000U)
Defines a negative zero value for the half data type.
#define CUDART_ONE_FP16 __ushort_as_half((unsigned short)0x3C00U)
Defines a value of 1.0 for the \texttt{half} data type.

#define CUDART_ZERO_FP16 __ushort_as_half((unsigned short)0x0000U)
Defines a positive zero value for the \texttt{half} data type.

1.2.2. Half Arithmetric Functions

Half Precision Intrinsics

To use these functions, include the header file \texttt{cuda_fp16.h} in your program.

\texttt{__host\_\_device\_\_habs (const __half a)}
Calculates the absolute value of input \texttt{half} number and returns the result.

Parameters

\texttt{a}
- half. Is only being read.

Returns

half
- The absolute value of \texttt{a}.

Description

Calculates the absolute value of input \texttt{half} number and returns the result.

\texttt{__host\_\_device\_\_hadd (const __half a, const __half b)}
Performs \texttt{half} addition in round-to-nearest-even mode.

Description

Performs \texttt{half} addition of inputs \texttt{a} and \texttt{b}, in round-to-nearest-even mode.
__host__  device__ hadd_rn (const __half a, const __half b)
Performs half addition in round-to-nearest-even mode.

Description
Performs half addition of inputs a and b, in round-to-nearest-even mode. Prevents floating-point contractions of mul+add into fma.

__host__  device__ hadd_sat (const __half a, const __half b)
Performs half addition in round-to-nearest-even mode, with saturation to [0.0, 1.0].

Parameters
a
- half. Is only being read.
b
- half. Is only being read.

Returns
half
- The sum of a and b, with respect to saturation.

Description
Performs half add of inputs a and b, in round-to-nearest-even mode, and clamps the result to range [0.0, 1.0]. NaN results are flushed to +0.0.

__host__  device__ hdiv (const __half a, const __half b)
Performs half division in round-to-nearest-even mode.

Description
Divides half input a by input b in round-to-nearest-even mode.
__device__ hfma (const __half a, const __half b, const __half c)
Performs half fused multiply-add in round-to-nearest-even mode.

Description
Performs half multiply on inputs a and b, then performs a half add of the result with c, rounding the result once in round-to-nearest-even mode.

__device__ hfma_relu (const __half a, const __half b, const __half c)
Performs half fused multiply-add in round-to-nearest-even mode with relu saturation.

Parameters
a
- half. Is only being read.
b
- half. Is only being read.
c
- half. Is only being read.

Returns
half
- The result of fused multiply-add operation on a, b, and c with relu saturation.

Description
Performs half multiply on inputs a and b, then performs a half add of the result with c, rounding the result once in round-to-nearest-even mode. Then negative result is clamped to 0. NaN result is converted to canonical NaN.

__device__ hfma_sat (const __half a, const __half b, const __half c)
Performs half fused multiply-add in round-to-nearest-even mode, with saturation to [0.0, 1.0].

Parameters
a
- half. Is only being read.
b
- half. Is only being read.
c
- half. Is only being read.

Returns

half

- The result of fused multiply-add operation on a, b, and c, with respect to saturation.

Description

Performs half multiply on inputs a and b, then performs a half add of the result with c, rounding the result once in round-to-nearest-even mode, and clamps the result to range [0.0, 1.0]. NaN results are flushed to +0.0.

__host__ __device__ hmul (const __half a, const __half b)

Performs half multiplication in round-to-nearest-even mode.

Description

Performs half multiplication of inputs a and b, in round-to-nearest-even mode.

__host__ __device__ hmul_rn (const __half a, const __half b)

Performs half multiplication in round-to-nearest-even mode.

Description

Performs half multiplication of inputs a and b, in round-to-nearest-even mode. Prevents floating-point contractions of mul+add or sub into fma.

__host__ __device__ hmul_sat (const __half a, const __half b)

Performs half multiplication in round-to-nearest-even mode, with saturation to [0.0, 1.0].

Parameters

a
- half. Is only being read.

b
- half. Is only being read.

Returns

half

- The result of multiplying a and b, with respect to saturation.
Description
Performs \texttt{half} multiplication of inputs \texttt{a} and \texttt{b}, in round-to-nearest-even mode, and clamps the result to range \([0.0, 1.0]\). NaN results are flushed to \(+0.0\).

\texttt{__host\_\_device\_\_hneg (const __half a)}

Negates input \texttt{half} number and returns the result.

\texttt{__host\_\_device\_\_hsub (const __half a, const __half b)}

Performs \texttt{half} subtraction in round-to-nearest-even mode.

\texttt{__host\_\_device\_\_hsub\_rn (const __half a, const __half b)}

Performs \texttt{half} subtraction in round-to-nearest-even mode. Prevents floating-point contractions of \texttt{mul+sub} into \texttt{fma}.

\texttt{__host\_\_device\_\_hsub\_sat (const __half a, const __half b)}

Performs \texttt{half} subtraction in round-to-nearest-even mode, with saturation to \([0.0, 1.0]\).

Parameters
\begin{itemize}
  \item \texttt{a}
    - \texttt{half}. Is only being read.
  \item \texttt{b}
    - \texttt{half}. Is only being read.
\end{itemize}

Returns
\begin{itemize}
  \item \texttt{half}
    \begin{itemize}
      \item The result of subtraction of \texttt{b} from \texttt{a}, with respect to saturation.
    \end{itemize}
\end{itemize}
Description

Subtracts half input b from input a in round-to-nearest-even mode, and clamps the result to range [0.0, 1.0]. NaN results are flushed to +0.0.

__device__ atomicAdd (const __half *address, const __half val)

Adds val to the value stored at address in global or shared memory, and writes this value back to address. This operation is performed in one atomic operation.

Parameters

address
- half*. An address in global or shared memory.

val
- half. The value to be added.

Returns

half
  The old value read from address.

Description

The location of address must be in global or shared memory. This operation has undefined behavior otherwise. This operation is only supported by devices of compute capability 7.x and higher.

Note:

For more details for this function see the Atomic Functions section in the CUDA C++ Programming Guide.

__host____device__ operator* (const __half lh, const __half rh)

Description

Performs half multiplication operation. See also __hmul(__half, __half)
**__host____device__operator*= (__half lh, const __half rh)**

**Description**
Performs half compound assignment with multiplication operation.

**__host____device__operator+ (const __half h)**

**Description**
Implements half unary plus operator, returns input value.

**__host____device__operator+ (const __half lh, const __half rh)**

**Description**
Performs half addition operation. See also __hadd(__half, __half)

**__host____device__operator++ (__half h, const int ignored)**

**Description**
Performs half postfix increment operation.

**__host____device__operator++ (__half h)**

**Description**
Performs half prefix increment operation.

**__host____device__operator+= (__half lh, const __half rh)**

**Description**
Performs half compound assignment with addition operation.

**__host____device__operator- (const __half h)**

**Description**
Implements half unary minus operator. See also __hneg(__half)
__host____device__operator- (const __half lh, const __half rh)

Description
Performs half subtraction operation. See also __hsubf __half, __half

__host____device__operator-- (__half h, const int ignored)

Description
Performs half postfix decrement operation.

__host____device__operator-- (__half h)

Description
Performs half prefix decrement operation.

__host____device__operator-= (const __half rh)

Description
Performs half compound assignment with subtraction operation.

__host____device__operator/ (const __half lh, const __half rh)

Description
Performs half division operation. See also __hdivf __half, __half

__host____device__operator/= (__half lh, const __half rh)

Description
Performs half compound assignment with division operation.

1.2.3. Half2 Arithmetic Functions

Half Precision Intrinsics

To use these functions, include the header file cuda_fp16.h in your program.
__host__ __device__ h2div (const __half2 a, const __half2 b)
Performs half2 vector division in round-to-nearest-even mode.

Description
Divides half2 input vector a by input vector b in round-to-nearest-even mode.

__host__ __device__ habs2 (const __half2 a)
Calculates the absolute value of both halves of the input half2 number and returns the result.

Parameters
a
- half2. Is only being read.

Returns
half2
- Returns a with the absolute value of both halves.

Description
Calculates the absolute value of both halves of the input half2 number and returns the result.

__host__ __device__ hadd2 (const __half2 a, const __half2 b)
Performs half2 vector addition in round-to-nearest-even mode.

Description
Performs half2 vector add of inputs a and b, in round-to-nearest-even mode.

__host__ __device__ hadd2_rn (const __half2 a, const __half2 b)
Performs half2 vector addition in round-to-nearest-even mode.

Description
Performs half2 vector add of inputs a and b, in round-to-nearest-even mode. Prevents floating-point contractions of mul+add into fma.
__host__ __device__ hadd2_sat (const __half2 a, const __half2 b)
Performs half2 vector addition in round-to-nearest-even mode, with saturation to [0.0, 1.0].

Parameters
a
- half2. Is only being read.
b
- half2. Is only being read.

Returns
half2
► The sum of a and b, with respect to saturation.

Description
Performs half2 vector add of inputs a and b, in round-to-nearest-even mode, and clamps the results to range [0.0, 1.0]. NaN results are flushed to +0.0.

__device__ hcmadd (const __half2 a, const __half2 b, const __half2 c)
Performs fast complex multiply-accumulate.

Parameters
a
- half2. Is only being read.
b
- half2. Is only being read.
c
- half2. Is only being read.

Returns
half2
► The result of complex multiply-accumulate operation on complex numbers a, b, and c

Description
Interprets vector half2 input pairs a, b, and c as complex numbers in half precision and performs complex multiply-accumulate operation: a*b + c
__device____hfma2 (const __half2 a, const __half2 b, const __half2 c)
Performs half2 vector fused multiply-add in round-to-nearest-even mode.

Description
Performs half2 vector multiply on inputs a and b, then performs a half2 vector add of the result with c, rounding the result once in round-to-nearest-even mode.

__device____hfma2_relu (const __half2 a, const __half2 b, const __half2 c)
Performs half2 vector fused multiply-add in round-to-nearest-even mode with relu saturation.

Parameters
a
  - half2. Is only being read.
b
  - half2. Is only being read.
c
  - half2. Is only being read.

Returns
half2
  - The result of elementwise fused multiply-add operation on vectors a, b, and c with relu saturation.

Description
Performs half2 vector multiply on inputs a and b, then performs a half2 vector add of the result with c, rounding the result once in round-to-nearest-even mode. Then negative result is clamped to 0. NaN result is converted to canonical NaN.

__device____hfma2_sat (const __half2 a, const __half2 b, const __half2 c)
Performs half2 vector fused multiply-add in round-to-nearest-even mode, with saturation to [0.0, 1.0].

Parameters
a
  - half2. Is only being read.
b
  - half2. Is only being read.

c
  - half2. Is only being read.

**Returns**

half2

- The result of elementwise fused multiply-add operation on vectors a, b, and c, with respect to saturation.

**Description**

Performs half2 vector multiply on inputs a and b, then performs a half2 vector add of the result with c, rounding the result once in round-to-nearest-even mode, and clamps the results to range [0.0, 1.0]. NaN results are flushed to +0.0.

```c
__host__ __device__ hmul2 (const __half2 a, const __half2 b)
```

Performs half2 vector multiplication in round-to-nearest-even mode.

**Description**

Performs half2 vector multiplication of inputs a and b, in round-to-nearest-even mode.

```c
__host__ __device__ hmul2_rn (const __half2 a, const __half2 b)
```

Performs half2 vector multiplication in round-to-nearest-even mode.

**Description**

Performs half2 vector multiplication of inputs a and b, in round-to-nearest-even mode. Prevents floating-point contractions of mul+add or sub into fma.

```c
__host__ __device__ hmul2_sat (const __half2 a, const __half2 b)
```

Performs half2 vector multiplication in round-to-nearest-even mode, with saturation to [0.0, 1.0].

**Parameters**

**a**

- half2. Is only being read.
b
- half2. Is only being read.

**Returns**
half2

- The result of elementwise multiplication of vectors a and b, with respect to saturation.

**Description**
Performs half2 vector multiplication of inputs a and b, in round-to-nearest-even mode, and clamps the results to range [0.0, 1.0]. NaN results are flushed to +0.0.

```c
__host__ __device__ hmul2 (const __half2 a, const __half2 b)
```

Negates both halves of the input half2 number and returns the result.

**Description**
Negates both halves of the input half2 number a and returns the result.

```c
__host__ __device__ hneg2 (const __half2 a)
```

Performs half2 vector subtraction in round-to-nearest-even mode.

**Description**
Subtracts half2 input vector b from input vector a in round-to-nearest-even mode.

```c
__host__ __device__ hsub2 (const __half2 a, const __half2 b)
```

Performs half2 vector subtraction in round-to-nearest-even mode.

**Description**
Subtracts half2 input vector b from input vector a in round-to-nearest-even mode. Prevents floating-point contractions of mul+sub into fma.
__host__ __device__ hsub2_sat (const __half2 a, const __half2 b)
Performs half2 vector subtraction in round-to-nearest-even mode, with saturation to [0.0, 1.0].

Parameters
a
- half2. Is only being read.
b
- half2. Is only being read.

Returns
half2

- The subtraction of vector b from a, with respect to saturation.

Description
Subtracts half2 input vector b from input vector a in round-to-nearest-even mode, and clamps the results to range [0.0, 1.0]. NaN results are flushed to +0.0.

__device__ atomicAdd (const __half2 *address, const __half2 val)
Vector add val to the value stored at address in global or shared memory, and writes this value back to address. The atomicity of the add operation is guaranteed separately for each of the two __half elements; the entire __half2 is not guaranteed to be atomic as a single 32-bit access.

Parameters
address
- half2*. An address in global or shared memory.
val
- half2. The value to be added.

Returns
half2

- The old value read from address.
Description
The location of address must be in global or shared memory. This operation has undefined behavior otherwise. This operation is natively supported by devices of compute capability 6.x and higher, older devices use emulation path.

Note:
For more details for this function see the Atomic Functions section in the CUDA C++ Programming Guide.

__host__ __device__ operator* (const __half2 lh, const __half2 rh)

Description
Performs packed half multiplication operation. See also __hmul2(__half2, __half2)

__host__ __device__ operator*=(__half2 lh, const __half2 rh)

Description
Performs packed half compound assignment with multiplication operation.

__host__ __device__ operator+ (const __half2 h)

Description
Implements packed half unary plus operator, returns input value.

__host__ __device__ operator+ (const __half2 lh, const __half2 rh)

Description
Performs packed half addition operation. See also __hadd2(__half2, __half2)

__host__ __device__ operator++ (__half2 h, const int ignored)

Description
Performs packed half postfix increment operation.
__host__ __device__ operator++ (__half2 h)

Description
Performs packed half prefix increment operation.

__host__ __device__ operator+= (__half2 lh, const __half2 rh)

Description
Performs packed half compound assignment with addition operation.

__host__ __device__ operator- (const __half2 h)

Description
Implements packed half unary minus operator. See also __hneg2(__half2)

__host__ __device__ operator- (const __half2 lh, const __half2 rh)

Description
Performs packed half subtraction operation. See also __hsub2(__half2, __half2)

__host__ __device__ operator-- (const __half2 h, const int ignored)

Description
Performs packed half postfix decrement operation.

__host__ __device__ operator-- (const __half2 h)

Description
Performs packed half prefix decrement operation.

__host__ __device__ operator-= (__half2 lh, const __half2 rh)

Description
Performs packed half compound assignment with subtraction operation.
**__host__** **__device__** **operator/** (const **__half2** lh, const **__half2** rh)

**Description**
Performs packed **half** division operation. See also **_h2div([half2, half2])**

**__host__** **__device__** **operator/**= (**__half2** lh, const **__half2** rh)

**Description**
Performs packed **half** compound assignment with division operation.

### 1.2.4. Half Comparison Functions

**Half Precision Intrinsics**

To use these functions, include the header file **cuda_fp16.h** in your program.

**__host__** **__device__** **bool** **__heq** (const **__half** a, const **__half** b)

Performs **half** if-equal comparison.

**Parameters**

- **a**
  - **half**. Is only being read.

- **b**
  - **half**. Is only being read.

**Returns**

- **bool**
  - The boolean result of if-equal comparison of **a** and **b**.

**Description**
Performs **half** if-equal comparison of inputs **a** and **b**. **NaN** inputs generate false results.
__host____device__ bool __hequ (const __half a, const __half b)
Performs \texttt{half} unordered if-equal comparison.

**Parameters**

- **a**
  - \texttt{half}. Is only being read.
- **b**
  - \texttt{half}. Is only being read.

**Returns**

- \texttt{bool}
  - The boolean result of unordered if-equal comparison of \texttt{a} and \texttt{b}.

**Description**

Performs \texttt{half} if-equal comparison of inputs \texttt{a} and \texttt{b}. NaN inputs generate true results.

__host____device__ bool __hge (const __half a, const __half b)
Performs \texttt{half} greater-equal comparison.

**Parameters**

- **a**
  - \texttt{half}. Is only being read.
- **b**
  - \texttt{half}. Is only being read.

**Returns**

- \texttt{bool}
  - The boolean result of greater-equal comparison of \texttt{a} and \texttt{b}.

**Description**

Performs \texttt{half} greater-equal comparison of inputs \texttt{a} and \texttt{b}. NaN inputs generate false results.
__host__ __device__ bool __hgeu (const __half a, const __half b)
Performs \texttt{half} unordered greater-equal comparison.

\textbf{Parameters}

\texttt{a}
- half. Is only being read.

\texttt{b}
- half. Is only being read.

\textbf{Returns}

\texttt{bool}
- The boolean result of unordered greater-equal comparison of \texttt{a} and \texttt{b}.

\textbf{Description}

Performs \texttt{half} greater-equal comparison of inputs \texttt{a} and \texttt{b}. NaN inputs generate true results.

__host__ __device__ bool __hgt (const __half a, const __half b)
Performs \texttt{half} greater-than comparison.

\textbf{Parameters}

\texttt{a}
- half. Is only being read.

\texttt{b}
- half. Is only being read.

\textbf{Returns}

\texttt{bool}
- The boolean result of greater-than comparison of \texttt{a} and \texttt{b}.

\textbf{Description}

Performs \texttt{half} greater-than comparison of inputs \texttt{a} and \texttt{b}. NaN inputs generate false results.
__host____device__ bool __hgtu (const __half a, const __half b)
Performs half unordered greater-than comparison.

Parameters
a
- half. Is only being read.
b
- half. Is only being read.

Returns
bool
  ▶ The boolean result of unordered greater-than comparison of a and b.

Description
Performs half greater-than comparison of inputs a and b. NaN inputs generate true results.

__host____device__ int __hisinf (const __half a)
Checks if the input half number is infinite.

Parameters
a
- half. Is only being read.

Returns
int
  ▶ -1 iff a is equal to negative infinity,
  ▶ 1 iff a is equal to positive infinity,
  ▶ 0 otherwise.

Description
Checks if the input half number a is infinite.
__host__ __device__ bool __hisnan (const __half a)
Determine whether half argument is a NaN.

Parameters
a
- half. Is only being read.

Returns
bool
- true iff argument is NaN.

Description
Determine whether half value a is a NaN.

__host__ __device__ bool __hle (const __half a, const __half b)
Performs half less-equal comparison.

Parameters
a
- half. Is only being read.
b
- half. Is only being read.

Returns
bool
- The boolean result of less-equal comparison of a and b.

Description
Performs half less-equal comparison of inputs a and b. NaN inputs generate false results.

__host__ __device__ bool __hleu (const __half a, const __half b)
Performs half unordered less-equal comparison.

Parameters
a
- half. Is only being read.
b
- half. Is only being read.

Returns
bool

- The boolean result of unordered less-equal comparison of a and b.

Description
Performs half less-equal comparison of inputs a and b. NaN inputs generate true results.

__host__ __device__ bool __hlt (const __half a, const __half b)
Performs half less-than comparison.

Parameters
a
- half. Is only being read.

b
- half. Is only being read.

Returns
bool

- The boolean result of less-than comparison of a and b.

Description
Performs half less-than comparison of inputs a and b. NaN inputs generate false results.

__host__ __device__ bool __hltu (const __half a, const __half b)
Performs half unordered less-than comparison.

Parameters
a
- half. Is only being read.

b
- half. Is only being read.

Returns
bool
The boolean result of unordered less-than comparison of $a$ and $b$.

**Description**

Performs half less-than comparison of inputs $a$ and $b$. NaN inputs generate true results.

```c
__host__ __device__ hmax (const __half a, const __half b)
```

Calculates half maximum of two input values.

**Description**

Calculates half $\max(a, b)$ defined as $(a > b) ? a : b$.

- If either of inputs is NaN, the other input is returned.
- If both inputs are NaNs, then canonical NaN is returned.
- If values of both inputs are 0.0, then $+0.0 > -0.0$

```c
__host__ __device__ hmax_nan (const __half a, const __half b)
```

Calculates half maximum of two input values, NaNs pass through.

**Description**

Calculates half $\max(a, b)$ defined as $(a > b) ? a : b$.

- If either of inputs is NaN, then canonical NaN is returned.
- If values of both inputs are 0.0, then $+0.0 > -0.0$

```c
__host__ __device__ hmin (const __half a, const __half b)
```

Calculates half minimum of two input values.

**Description**

Calculates half $\min(a, b)$ defined as $(a < b) ? a : b$.

- If either of inputs is NaN, the other input is returned.
- If both inputs are NaNs, then canonical NaN is returned.
- If values of both inputs are 0.0, then $+0.0 > -0.0$
__host__ __device__ hmin_nan (const __half a, const __half b)
Calculates half minimum of two input values, NaNs pass through.

Description
Calculates half min(a, b) defined as (a < b) ? a : b.

- If either of inputs is NaN, then canonical NaN is returned.
- If values of both inputs are 0.0, then +0.0 > -0.0

__host__ __device__ bool __hne (const __half a, const __half b)
Performs half not-equal comparison.

Parameters
a
- half. Is only being read.

b
- half. Is only being read.

Returns
bool
- The boolean result of not-equal comparison of a and b.

Description
Performs half not-equal comparison of inputs a and b. NaN inputs generate false results.

__host__ __device__ bool __hneu (const __half a, const __half b)
Performs half unordered not-equal comparison.

Parameters
a
- half. Is only being read.

b
- half. Is only being read.
Returns

bool

- The boolean result of unordered not-equal comparison of \( a \) and \( b \).

Description

Performs half not-equal comparison of inputs \( a \) and \( b \). NaN inputs generate true results.

\[
_{\text{host}}\_{\text{device}}\_{\text{CUDA_FP16}}\_{\text{FORCEINLINE}}\text{ bool operator}!=(\text{const }\text{half }lh, \text{const }\text{half }rh)
\]

Description

Performs half unordered compare not-equal operation. See also \( _{\text{hneu}}(\text{half}, \text{half}) \)

\[
_{\text{host}}\_{\text{device}}\_{\text{CUDA_FP16}}\_{\text{FORCEINLINE}}\text{ bool operator}< (\text{const }\text{half }lh, \text{const }\text{half }rh)
\]

Description

Performs half ordered less-than compare operation. See also \( _{\text{hlt}}(\text{half}, \text{half}) \)

\[
_{\text{host}}\_{\text{device}}\_{\text{CUDA_FP16}}\_{\text{FORCEINLINE}}\text{ bool operator}<= (\text{const }\text{half }lh, \text{const }\text{half }rh)
\]

Description

Performs half ordered less-or-equal compare operation. See also \( _{\text{hle}}(\text{half}, \text{half}) \)

\[
_{\text{host}}\_{\text{device}}\_{\text{CUDA_FP16}}\_{\text{FORCEINLINE}}\text{ bool operator}== (\text{const }\text{half }lh, \text{const }\text{half }rh)
\]

Description

Performs half ordered compare equal operation. See also \( _{\text{heq}}(\text{half}, \text{half}) \)
__host__ __device__ __CUDA_FP16_FORCEINLINE__ bool operator> (const __half lh, const __half rh)

Description
Performs \texttt{half} ordered greater-than compare operation. See also \_\_\_\_\_hgt\_\_\_\_half, \_\_\_\_\_half\_

__host__ __device__ __CUDA_FP16_FORCEINLINE__ bool operator>=(const __half lh, const __half rh)

Description
Performs \texttt{half} ordered greater-or-equal compare operation. See also \_\_\_\_\_hge\_\_\_\_half, \_\_\_\_\_half\_

### 1.2.5. Half2 Comparison Functions

#### Half Precision Intrinsics

To use these functions, include the header file \texttt{cuda_fp16.h} in your program.

__host__ __device__ bool \_\_\_\_\_hbeq2\_\_\_\_half2 \_\_\_\_\_half2 a, const __half2 b

Performs \texttt{half2} vector if-equal comparison and returns boolean true iff both \texttt{half} results are true, boolean false otherwise.

**Parameters**

- \textbf{a}
  - half2. Is only being read.
- \textbf{b}
  - half2. Is only being read.

**Returns**

bool
- true if both \texttt{half} results of if-equal comparison of vectors \texttt{a} and \texttt{b} are true;
- false otherwise.

**Description**

Performs \texttt{half2} vector if-equal comparison of inputs \texttt{a} and \texttt{b}. The bool result is set to true only if both \texttt{half} if-equal comparisons evaluate to true, or false otherwise. \texttt{NaN} inputs generate false results.
__host__ __device__ bool __hbequ2 (const __half2 a, const __half2 b)
Performs half2 vector unordered if-equal comparison and returns boolean true iff both half results are true, boolean false otherwise.

Parameters

a
- half2. Is only being read.

b
- half2. Is only being read.

Returns

bool

- true if both half results of unordered if-equal comparison of vectors a and b are true;
- false otherwise.

Description

Performs half2 vector if-equal comparison of inputs a and b. The bool result is set to true only if both half if-equal comparisons evaluate to true, or false otherwise. NaN inputs generate true results.

__host__ __device__ bool __hbge2 (const __half2 a, const __half2 b)
Performs half2 vector greater-equal comparison and returns boolean true iff both half results are true, boolean false otherwise.

Parameters

a
- half2. Is only being read.

b
- half2. Is only being read.

Returns

bool

- true if both half results of greater-equal comparison of vectors a and b are true;
- false otherwise.
**Description**
Performs `half2` vector greater-equal comparison of inputs `a` and `b`. The bool result is set to true only if both `half` greater-equal comparisons evaluate to true, or false otherwise. NaN inputs generate false results.

```c
__host__ __device__ bool __hbgeu2 (const __half2 a, const __half2 b)
```
Performs `half2` vector unordered greater-equal comparison and returns boolean true iff both `half` results are true, boolean false otherwise.

**Parameters**
- `a` - `half2`. Is only being read.
- `b` - `half2`. Is only being read.

**Returns**
- bool
  - true if both `half` results of unordered greater-equal comparison of vectors `a` and `b` are true;
  - false otherwise.

**Description**
Performs `half2` vector greater-than comparison and returns boolean true iff both `half` results are true, boolean false otherwise.

```c
__host__ __device__ bool __hbgt2 (const __half2 a, const __half2 b)
```
Performs `half2` vector greater-than comparison and returns boolean true iff both `half` results are true, boolean false otherwise.

**Parameters**
- `a` - `half2`. Is only being read.
- `b` - `half2`. Is only being read.
Returns
bool
- true if both half results of greater-than comparison of vectors a and b are true;
- false otherwise.

Description
Performs half2 vector greater-than comparison of inputs a and b. The bool result is set to true only if both half greater-than comparisons evaluate to true, or false otherwise. NaN inputs generate false results.

__host__ __device__ bool __hbgtu2 (const __half2 a, const __half2 b)
Performs half2 vector unordered greater-than comparison and returns boolean true iff both half results are true, boolean false otherwise.

Parameters
a
- half2. Is only being read.

b
- half2. Is only being read.

Returns
bool
- true if both half results of unordered greater-than comparison of vectors a and b are true;
- false otherwise.

Description
Performs half2 vector greater-than comparison of inputs a and b. The bool result is set to true only if both half greater-than comparisons evaluate to true, or false otherwise. NaN inputs generate true results.
__host__ __device__ bool __hble2 (const __half2 a, const __half2 b)
Performs half2 vector less-equal comparison and returns boolean true iff both half results are true, boolean false otherwise.

Parameters
a
- half2. Is only being read.
b
- half2. Is only being read.

Returns
bool
  ▶ true if both half results of less-equal comparison of vectors a and b are true;
  ▶ false otherwise.

Description
Performs half2 vector less-equal comparison of inputs a and b. The bool result is set to true only if both half less-equal comparisons evaluate to true, or false otherwise. NaN inputs generate false results.

__host__ __device__ bool __hbleu2 (const __half2 a, const __half2 b)
Performs half2 vector unordered less-equal comparison and returns boolean true iff both half results are true, boolean false otherwise.

Parameters
a
- half2. Is only being read.
b
- half2. Is only being read.

Returns
bool
  ▶ true if both half results of unordered less-equal comparison of vectors a and b are true;
  ▶ false otherwise.
Description
Performs half2 vector less-equal comparison of inputs a and b. The bool result is set to true only if both half less-equal comparisons evaluate to true, or false otherwise. NaN inputs generate true results.

__host__ __device__ bool __hblt2 (const __half2 a, const __half2 b)
Performs half2 vector less-than comparison and returns boolean true iff both half results are true, boolean false otherwise.

Parameters
a
- half2. Is only being read.
b
- half2. Is only being read.

Returns
bool
➤ true if both half results of less-than comparison of vectors a and b are true;
➤ false otherwise.

Description
Performs half2 vector less-than comparison of inputs a and b. The bool result is set to true only if both half less-than comparisons evaluate to true, or false otherwise. NaN inputs generate false results.

__host__ __device__ bool __hbltu2 (const __half2 a, const __half2 b)
Performs half2 vector unordered less-than comparison and returns boolean true iff both half results are true, boolean false otherwise.

Parameters
a
- half2. Is only being read.
b
- half2. Is only being read.
Returns

bool

- true if both half results of unordered less-than comparison of vectors a and b are true;
- false otherwise.

Description

Performs half2 vector less-than comparison of inputs a and b. The bool result is set to true only if both half less-than comparisons evaluate to true, or false otherwise. NaN inputs generate true results.

__host__ __device__ bool __hbne2 (const __half2 a, const __half2 b)

Performs half2 vector not-equal comparison and returns boolean true iff both half results are true, boolean false otherwise.

Parameters

a
- half2. Is only being read.

b
- half2. Is only being read.

Returns

bool

- true if both half results of not-equal comparison of vectors a and b are true,
- false otherwise.

Description

Performs half2 vector not-equal comparison of inputs a and b. The bool result is set to true only if both half not-equal comparisons evaluate to true, or false otherwise. NaN inputs generate false results.
__host____device__ bool __hbneu2 (const __half2 a, const __half2 b)

Performs half2 vector unordered not-equal comparison and returns boolean true iff both half results are true, boolean false otherwise.

Parameters

a
- half2. Is only being read.

b
- half2. Is only being read.

Returns

bool

- true if both half results of unordered not-equal comparison of vectors a and b are true;
- false otherwise.

Description

Performs half2 vector not-equal comparison of inputs a and b. The bool result is set to true only if both half not-equal comparisons evaluate to true, or false otherwise. NaN inputs generate true results.

__host____device____heq2 (const __half2 a, const __half2 b)

Performs half2 vector if-equal comparison.

Parameters

a
- half2. Is only being read.

b
- half2. Is only being read.

Returns

half2

- The vector result of if-equal comparison of vectors a and b.

Description

Performs half2 vector if-equal comparison of inputs a and b. The corresponding half results are set to 1.0 for true, or 0.0 for false. NaN inputs generate false results.
__host____device__ unsigned __heq2_mask (const __half2 a, const __half2 b)
Performs half2 vector if-equal comparison.

Parameters
a
- half2. Is only being read.
b
- half2. Is only being read.

Returns
unsigned int
▶ The vector mask result of if-equal comparison of vectors a and b.

Description
Performs half2 vector if-equal comparison of inputs a and b. The corresponding unsigned bits are set to 0xFFFF for true, or 0x0 for false. NaN inputs generate false results.

__host____device____hequ2 (const __half2 a, const __half2 b)
Performs half2 vector unordered if-equal comparison.

Parameters
a
- half2. Is only being read.
b
- half2. Is only being read.

Returns
half2
▶ The vector result of unordered if-equal comparison of vectors a and b.

Description
Performs half2 vector if-equal comparison of inputs a and b. The corresponding half results are set to 1.0 for true, or 0.0 for false. NaN inputs generate true results.
__host__ __device__ unsigned __hequ2_mask (const __half2 a, const __half2 b)
Performs `half2` vector unordered if-equal comparison.

Parameters
a  
- `half2`. Is only being read.

b  
- `half2`. Is only being read.

Returns
unsigned int
  ▶ The vector mask result of unordered if-equal comparison of vectors a and b.

Description
Performs `half2` vector if-equal comparison of inputs a and b. The corresponding unsigned bits are set to 0xFFFF for true, or 0x0 for false. NaN inputs generate true results.

__host__ __device__ hge2 (const __half2 a, const __half2 b)
Performs `half2` vector greater-equal comparison.

Parameters
a  
- `half2`. Is only being read.

b  
- `half2`. Is only being read.

Returns
`half2`
  ▶ The vector result of greater-equal comparison of vectors a and b.

Description
Performs `half2` vector greater-equal comparison of inputs a and b. The corresponding `half` results are set to 1.0 for true, or 0.0 for false. NaN inputs generate false results.
__host____device__ unsigned __hge2_mask (const __half2 a, const __half2 b)
Performs half2 vector greater-equal comparison.

Parameters
a
- half2. Is only being read.
b
- half2. Is only being read.

Returns
unsigned int
- The vector mask result of greater-equal comparison of vectors a and b.

Description
Performs half2 vector greater-equal comparison of inputs a and b. The corresponding unsigned bits are set to 0xFFFF for true, or 0x0 for false. NaN inputs generate false results.

__host____device____hgeu2 (const __half2 a, const __half2 b)
Performs half2 vector unordered greater-equal comparison.

Parameters
a
- half2. Is only being read.
b
- half2. Is only being read.

Returns
half2
- The half2 vector result of unordered greater-equal comparison of vectors a and b.

Description
Performs half2 vector greater-equal comparison of inputs a and b. The corresponding half results are set to 1.0 for true, or 0.0 for false. NaN inputs generate true results.
__host____device__ unsigned __hgeu2_mask (const __half2 a, const __half2 b)
Performs half2 vector unordered greater-equal comparison.

Parameters
a
- half2. Is only being read.
b
- half2. Is only being read.

Returns
unsigned int
- The vector mask result of unordered greater-equal comparison of vectors a and b.

Description
Performs half2 vector greater-equal comparison of inputs a and b. The corresponding unsigned bits are set to 0xFFFF for true, or 0x0 for false. NaN inputs generate true results.

__host____device____hgt2 (const __half2 a, const __half2 b)
Performs half2 vector greater-than comparison.

Parameters
a
- half2. Is only being read.
b
- half2. Is only being read.

Returns
half2
- The vector result of greater-than comparison of vectors a and b.

Description
Performs half2 vector greater-than comparison of inputs a and b. The corresponding half results are set to 1.0 for true, or 0.0 for false. NaN inputs generate false results.
__host__ __device__ unsigned __hgt2_mask (const __half2 a, const __half2 b)
Performs half2 vector greater-than comparison.

Parameters

a
  - half2. Is only being read.

b
  - half2. Is only being read.

Returns

unsigned int
  ▶ The vector mask result of greater-than comparison of vectors a and b.

Description

Performs half2 vector greater-than comparison of inputs a and b. The corresponding unsigned bits are set to 0xFFFF for true, or 0x0 for false. NaN inputs generate false results.

__host__ __device__ hgtu2 (const __half2 a, const __half2 b)
Performs half2 vector unordered greater-than comparison.

Parameters

a
  - half2. Is only being read.

b
  - half2. Is only being read.

Returns

half2
  ▶ The half2 vector result of unordered greater-than comparison of vectors a and b.

Description

Performs half2 vector greater-than comparison of inputs a and b. The corresponding half results are set to 1.0 for true, or 0.0 for false. NaN inputs generate true results.
__host__ device unsigned __hgtu2_mask (const __half2 a, const __half2 b)
Performs half2 vector unordered greater-than comparison.

Parameters
a
- half2. Is only being read.
b
- half2. Is only being read.

Returns
unsigned int
▶ The vector mask result of unordered greater-than comparison of vectors a and b.

Description
Performs half2 vector greater-than comparison of inputs a and b. The corresponding unsigned bits are set to 0xFFFF for true, or 0x0 for false. NaN inputs generate true results.

__host__ device hisnan2 (const __half2 a)
Determine whether half2 argument is a NaN.

Parameters
a
- half2. Is only being read.

Returns
half2
▶ The half2 with the corresponding half results set to 1.0 for NaN, 0.0 otherwise.

Description
Determine whether each half of input half2 number a is a NaN.

__host__ device hle2 (const __half2 a, const __half2 b)
Performs half2 vector less-equal comparison.

Parameters
a
- half2. Is only being read.
b
  - half2. Is only being read.

**Returns**

half2

- The half2 result of less-equal comparison of vectors a and b.

**Description**

Performs half2 vector less-equal comparison of inputs a and b. The corresponding half results are set to 1.0 for true, or 0.0 for false. NaN inputs generate false results.

```c
__host__ __device__ unsigned __hle2_mask (const __half2 a, const __half2 b)
```

Performs half2 vector less-equal comparison.

**Parameters**

a
  - half2. Is only being read.

b
  - half2. Is only being read.

**Returns**

unsigned int

- The vector mask result of less-equal comparison of vectors a and b.

**Description**

Performs half2 vector less-equal comparison of inputs a and b. The corresponding unsigned bits are set to 0xFFFF for true, or 0x0 for false. NaN inputs generate false results.

```c
__host__ __device__ __hleu2 (const __half2 a, const __half2 b)
```

Performs half2 vector unordered less-equal comparison.

**Parameters**

a
  - half2. Is only being read.

b
  - half2. Is only being read.
Returns

half2

- The vector result of unordered less-equal comparison of vectors a and b.

Description

Performs half2 vector less-equal comparison of inputs a and b. The corresponding half results are set to 1.0 for true, or 0.0 for false. NaN inputs generate true results.

__host__ device__ unsigned __hleu2_mask (const __half2 a, const __half2 b)

Performs half2 vector unordered less-equal comparison.

Parameters

a
- half2. Is only being read.

b
- half2. Is only being read.

Returns

unsigned int

- The vector mask result of unordered less-equal comparison of vectors a and b.

Description

Performs half2 vector less-equal comparison of inputs a and b. The corresponding unsigned bits are set to 0xFFFF for true, or 0x0 for false. NaN inputs generate true results.

__host__ device__ __half2 llt2 (const __half2 a, const __half2 b)

Performs half2 vector less-than comparison.

Parameters

a
- half2. Is only being read.

b
- half2. Is only being read.

Returns

half2

- The half2 vector result of less-than comparison of vectors a and b.
Description
Performs `half2` vector less-than comparison of inputs `a` and `b`. The corresponding `half` results are set to 1.0 for true, or 0.0 for false. NaN inputs generate false results.

```c
__host__ __device__ unsigned __hlt2_mask (const __half2 a, const __half2 b)
```
Performs `half2` vector less-than comparison.

Parameters
- `a` - `half2`. Is only being read.
- `b` - `half2`. Is only being read.

Returns
- `unsigned int` - The vector mask result of less-than comparison of vectors `a` and `b`.

Description
Performs `half2` vector less-than comparison of inputs `a` and `b`. The corresponding unsigned bits are set to 0xFFFF for true, or 0x0 for false. NaN inputs generate false results.

```c
__host__ __device__ __hltu2 (const __half2 a, const __half2 b)
```
Performs `half2` vector unordered less-than comparison.

Parameters
- `a` - `half2`. Is only being read.
- `b` - `half2`. Is only being read.

Returns
- `half2` - The vector result of unordered less-than comparison of vectors `a` and `b`. 
Description
Performs half2 vector less-than comparison of inputs \( a \) and \( b \). The corresponding half results are set to 1.0 for true, or 0.0 for false. NaN inputs generate true results.

__host__ __device__ unsigned __hltu2_mask (const __half2 \( a \), const __half2 \( b \))
Performs half2 vector unordered less-than comparison.

Parameters
\( a \)
- half2. Is only being read.
\( b \)
- half2. Is only being read.

Returns
unsigned int
- The vector mask result of unordered less-than comparison of vectors \( a \) and \( b \).

Description
Performs half2 vector less-than comparison of inputs \( a \) and \( b \). The corresponding unsigned bits are set to 0xFFFF for true, or 0x0 for false. NaN inputs generate true results.

__host__ __device__ hmax2 (const __half2 \( a \), const __half2 \( b \))
Calculates half2 vector maximum of two inputs.

Description
Calculates half2 vector max(\( a, b \)). Elementwise half operation is defined as \( (a > b) \ ? \ a : b \).
- If either of inputs is NaN, the other input is returned.
- If both inputs are NaNs, then canonical NaN is returned.
- If values of both inputs are 0.0, then +0.0 > -0.0
- The result of elementwise maximum of vectors \( a \) and \( b \)
__host__ __device__ hmax2_nan (const __half2 a, const __half2 b)
Calculates half2 vector maximum of two inputs, NaNs pass through.

Description
Calculates half2 vector max(a, b). Elementwise half operation is defined as (a > b) ? a : b.
- If either of inputs is NaN, then canonical NaN is returned.
- If values of both inputs are 0.0, then +0.0 > -0.0
- The result of elementwise maximum of vectors a and b, with NaNs pass through

__host__ __device__ hmin2 (const __half2 a, const __half2 b)
Calculates half2 vector minimum of two inputs.

Description
Calculates half2 vector min(a, b). Elementwise half operation is defined as (a < b) ? a : b.
- If either of inputs is NaN, the other input is returned.
- If both inputs are NaNs, then canonical NaN is returned.
- If values of both inputs are 0.0, then +0.0 > -0.0
- The result of elementwise minimum of vectors a and b

__host__ __device__ hmin2_nan (const __half2 a, const __half2 b)
Calculates half2 vector minimum of two inputs, NaNs pass through.

Description
Calculates half2 vector min(a, b). Elementwise half operation is defined as (a < b) ? a : b.
- If either of inputs is NaN, then canonical NaN is returned.
- If values of both inputs are 0.0, then +0.0 > -0.0
- The result of elementwise minimum of vectors a and b, with NaNs pass through
__host__ __device__ hne2 (const __half2 a, const __half2 b)
Performs half2 vector not-equal comparison.

Parameters

a
- half2. Is only being read.

b
- half2. Is only being read.

Returns

half2
  ▶ The vector result of not-equal comparison of vectors a and b.

Description

Performs half2 vector not-equal comparison of inputs a and b. The corresponding half results are set to 1.0 for true, or 0.0 for false. NaN inputs generate false results.

__host__ __device__ unsigned __hne2_mask (const __half2 a, const __half2 b)
Performs half2 vector not-equal comparison.

Parameters

a
- half2. Is only being read.

b
- half2. Is only being read.

Returns

unsigned int
  ▶ The vector mask result of not-equal comparison of vectors a and b.

Description

Performs half2 vector not-equal comparison of inputs a and b. The corresponding unsigned bits are set to 0xFFFF for true, or 0x0 for false. NaN inputs generate false results.
__host__ __device__ hneu2 (const __half2 a, const __half2 b)

Performs half2 vector unordered not-equal comparison.

**Parameters**

- `a` - half2. Is only being read.
- `b` - half2. Is only being read.

**Returns**

half2

- The vector result of unordered not-equal comparison of vectors `a` and `b`.

**Description**

Performs half2 vector not-equal comparison of inputs `a` and `b`. The corresponding half results are set to 1.0 for true, or 0.0 for false. NaN inputs generate true results.

__host__ __device__ unsigned __hneu2_mask (const __half2 a, const __half2 b)

Performs half2 vector unordered not-equal comparison.

**Parameters**

- `a` - half2. Is only being read.
- `b` - half2. Is only being read.

**Returns**

unsigned int

- The vector mask result of unordered not-equal comparison of vectors `a` and `b`.

**Description**

Performs half2 vector not-equal comparison of inputs `a` and `b`. The corresponding unsigned bits are set to 0xFFFF for true, or 0x0 for false. NaN inputs generate true results.
__host__ __device__ __CUDA_FP16_FORCEINLINE__ bool operator!= (const __half2 lh, const __half2 rh)

Description
Performs packed half unordered compare not-equal operation. See also __hbneu2(__half2, __half2)

__host__ __device__ __CUDA_FP16_FORCEINLINE__ bool operator< (const __half2 lh, const __half2 rh)

Description
Performs packed half ordered less-than compare operation. See also __hblt2(__half2, __half2)

__host__ __device__ __CUDA_FP16_FORCEINLINE__ bool operator<= (const __half2 lh, const __half2 rh)

Description
Performs packed half ordered less-or-equal compare operation. See also __hble2(__half2, __half2)

__host__ __device__ __CUDA_FP16_FORCEINLINE__ bool operator== (const __half2 lh, const __half2 rh)

Description
Performs packed half ordered compare equal operation. See also __hbeq2(__half2, __half2)

__host__ __device__ __CUDA_FP16_FORCEINLINE__ bool operator> (const __half2 lh, const __half2 rh)

Description
Performs packed half ordered greater-than compare operation. See also __hbgt2(__half2, __half2)
__host____device__ __CUDA_FP16_FORCEINLINE__ bool operator>=(const __half2 lh, const __half2 rh)

Description
Performs packed half ordered greater-or-equal compare operation. See also __hbge2(__half2, __half2)

1.2.6. Half Precision Conversion and Data Movement

Half Precision Intrinsics
To use these functions, include the header file cuda_fp16.h in your program.

__host____device____double2half (const double a)
Converts double number to half precision in round-to-nearest-even mode and returns half with converted value.

Parameters
a
- double. Is only being read.

Returns
half
  - a converted to half.

Description
Converts double number a to half precision in round-to-nearest-even mode.

__host____device____float22half2_rn (const float2 a)
Converts both components of float2 number to half precision in round-to-nearest-even mode and returns half2 with converted values.

Parameters
a
- float2. Is only being read.

Returns
half2
The `half2` which has corresponding halves equal to the converted `float2` components.

**Description**

Converts both components of `float2` to half precision in round-to-nearest-even mode and combines the results into one `half2` number. Low 16 bits of the return value correspond to `a.x` and high 16 bits of the return value correspond to `a.y`.

```c
__host__ __device__ float2half (const float a)
```

Converts float number to half precision in round-to-nearest-even mode and returns `half` with converted value.

**Parameters**

- `a` - float. Is only being read.

**Returns**

- `half` - `a` converted to half.

**Description**

Converts float number `a` to half precision in round-to-nearest-even mode.

```c
__host__ __device__ float2half2_rn (const float a)
```

Converts input to half precision in round-to-nearest-even mode and populates both halves of `half2` with converted value.

**Parameters**

- `a` - float. Is only being read.

**Returns**

- `half2` - The `half2` value with both halves equal to the converted half precision number.

**Description**

Converts input `a` to half precision in round-to-nearest-even mode and populates both halves of `half2` with converted value.
__host__ __device__ __float2half_rd (const __float a)
Converts float number to half precision in round-down mode and returns \texttt{half} with converted value.

**Parameters**

\texttt{a} 
- float. Is only being read.

**Returns**

\texttt{half} 
- \texttt{a} converted to half.

**Description**

Converts float number \texttt{a} to half precision in round-down mode.

__host__ __device__ __float2half_rn (const __float a)
Converts float number to half precision in round-to-nearest-even mode and returns \texttt{half} with converted value.

**Parameters**

\texttt{a} 
- float. Is only being read.

**Returns**

\texttt{half} 
- \texttt{a} converted to half.

**Description**

Converts float number \texttt{a} to half precision in round-to-nearest-even mode.

__host__ __device__ __float2half_ru (const __float a)
Converts float number to half precision in round-up mode and returns \texttt{half} with converted value.

**Parameters**

\texttt{a} 
- float. Is only being read.
Returns
half
▶ a converted to half.

Description
Converts float number a to half precision in round-up mode.

__host____device____float2half_rz (const float a)
Converts float number to half precision in round-towards-zero mode and returns half with converted value.

Parameters
a
- float. Is only being read.

Returns
half
▶ a converted to half.

Description
Converts float number a to half precision in round-towards-zero mode.

__host____device____floats2half2_rn (const float a, const float b)
Converts both input floats to half precision in round-to-nearest-even mode and returns half2 with converted values.

Parameters
a
- float. Is only being read.
b
- float. Is only being read.

Returns
half2
▶ The half2 value with corresponding halves equal to the converted input floats.
**Description**

Converts both input floats to half precision in round-to-nearest-even mode and combines the results into one `half2` number. Low 16 bits of the return value correspond to the input a, high 16 bits correspond to the input b.

```c
__host__ __device__ float2 __half22float2 (const __half2 a)
```

Converts both halves of `half2` to `float2` and returns the result.

**Parameters**

- `a` - `half2`. Is only being read.

**Returns**

- `float2` - a converted to `float2`.

**Description**

Converts both halves of `half2` input `a` to `float2` and returns the result.

```c
__host__ __device__ signed char __half2char_rz (const __half h)
```

Convert a half to a signed char in round-towards-zero mode.

**Parameters**

- `h` - `half`. Is only being read.

**Returns**

- `signed char` - h converted to a signed char.

**Description**

Convert the half-precision floating-point value `h` to a signed char integer in round-towards-zero mode. NaN inputs are converted to 0.
__host__ __device__ float __half2float (const __half a)
Converts half number to float.

Parameters
a
- float. Is only being read.

Returns
float
- a converted to float.

Description
Converts half number a to float.

__host__ __device__ __half2half2 (const __half a)
Returns half2 with both halves equal to the input value.

Parameters
a
- half. Is only being read.

Returns
half2
- The vector which has both its halves equal to the input a.

Description
Returns half2 number with both halves equal to the input a half number.

__device__ int __half2int_rd (const __half h)
Convert a half to a signed integer in round-down mode.

Parameters
h
- half. Is only being read.

Returns
int
- h converted to a signed integer.
Description
Convert the half-precision floating-point value $h$ to a signed integer in round-down mode. NaN inputs are converted to 0.

\texttt{\_device\_ \_ int \_ \_half2int\_rn (const \_ \_half \ h)}
Convert a half to a signed integer in round-to-nearest-even mode.

Parameters
\begin{itemize}
  \item \texttt{\_half} - half. Is only being read.
\end{itemize}

Returns
\begin{itemize}
  \item \texttt{\_int} - \texttt{\_half} converted to a signed integer.
\end{itemize}

Description
Convert the half-precision floating-point value $h$ to a signed integer in round-to-nearest-even mode. NaN inputs are converted to 0.

\texttt{\_device\_ \_ int \_ \_half2int\_ru (const \_ \_half \ h)}
Convert a half to a signed integer in round-up mode.

Parameters
\begin{itemize}
  \item \texttt{\_half} - half. Is only being read.
\end{itemize}

Returns
\begin{itemize}
  \item \texttt{\_int} - \texttt{\_half} converted to a signed integer.
\end{itemize}

Description
Convert the half-precision floating-point value $h$ to a signed integer in round-up mode. NaN inputs are converted to 0.
\_\_host\_\_\_device\_\_ int \_\_half2int\_rz (const \_\_half h)
Convert a half to a signed integer in round-towards-zero mode.

Parameters
h
- half. Is only being read.

Returns
int
▶ h converted to a signed integer.

Description
Convert the half-precision floating-point value h to a signed integer in round-towards-zero mode. NaN inputs are converted to 0.

\_\_device\_\_ long long int \_\_half2ll\_rd (const \_\_half h)
Convert a half to a signed 64-bit integer in round-down mode.

Parameters
h
- half. Is only being read.

Returns
long long int
▶ h converted to a signed 64-bit integer.

Description
Convert the half-precision floating-point value h to a signed 64-bit integer in round-down mode. NaN inputs return a long long int with hex value of 0x8000000000000000.

\_\_device\_\_ long long int \_\_half2ll\_rn (const \_\_half h)
Convert a half to a signed 64-bit integer in round-to-nearest-even mode.

Parameters
h
- half. Is only being read.
Returns
long long int

- h converted to a signed 64-bit integer.

Description
Convert the half-precision floating-point value h to a signed 64-bit integer in round-to-nearest-even mode. NaN inputs return a long long int with hex value of 0x8000000000000000.

__device__ long long int __half2ll_ru (const __half h)
Convert a half to a signed 64-bit integer in round-up mode.

Parameters
h
- half. Is only being read.

Returns
long long int

- h converted to a signed 64-bit integer.

Description
Convert the half-precision floating-point value h to a signed 64-bit integer in round-up mode. NaN inputs return a long long int with hex value of 0x8000000000000000.

__host__ __device__ long long int __half2ll_rz (const __half h)
Convert a half to a signed 64-bit integer in round-towards-zero mode.

Parameters
h
- half. Is only being read.

Returns
long long int

- h converted to a signed 64-bit integer.
**Description**

Convert the half-precision floating-point value \( h \) to a signed 64-bit integer in round-towards-zero mode. NaN inputs return a long long int with hex value of 0x8000000000000000.

```c
__device__ short int __half2short_rd (const __half h)
```

Convert a half to a signed short integer in round-down mode.

**Parameters**

- \( h \) - half. Is only being read.

**Returns**

- short int
  - \( h \) converted to a signed short integer.

**Description**

Convert the half-precision floating-point value \( h \) to a signed short integer in round-down mode. NaN inputs are converted to 0.

```c
__device__ short int __half2short_rn (const __half h)
```

Convert a half to a signed short integer in round-to-nearest-even mode.

**Parameters**

- \( h \) - half. Is only being read.

**Returns**

- short int
  - \( h \) converted to a signed short integer.

**Description**

Convert the half-precision floating-point value \( h \) to a signed short integer in round-to-nearest-even mode. NaN inputs are converted to 0.
__device__ short int __half2short_ru (const __half h)
Convert a half to a signed short integer in round-up mode.

Parameters
h
- half. Is only being read.

Returns
short int
▷ h converted to a signed short integer.

Description
Convert the half-precision floating-point value h to a signed short integer in round-up mode. NaN inputs are converted to 0.

__host____device__ short int __half2short_rz (const __half h)
Convert a half to a signed short integer in round-towards-zero mode.

Parameters
h
- half. Is only being read.

Returns
short int
▷ h converted to a signed short integer.

Description
Convert the half-precision floating-point value h to a signed short integer in round-towards-zero mode. NaN inputs are converted to 0.

__host____device__ unsigned char __half2uchar_rz (const __half h)
Convert a half to an unsigned char in round-towards-zero mode.

Parameters
h
- half. Is only being read.
Returns
unsigned char
  h converted to an unsigned char.

Description
Convert the half-precision floating-point value h to an unsigned char in round-towards-zero mode. NaN inputs are converted to 0.

__device__ unsigned int __half2uint_rd (const __half h)
Convert a half to an unsigned integer in round-down mode.

Parameters
h
  - half. Is only being read.

Returns
unsigned int
  h converted to an unsigned integer.

Description
Convert the half-precision floating-point value h to an unsigned integer in round-down mode. NaN inputs are converted to 0.

__device__ unsigned int __half2uint_rn (const __half h)
Convert a half to an unsigned integer in round-to-nearest-even mode.

Parameters
h
  - half. Is only being read.

Returns
unsigned int
  h converted to an unsigned integer.

Description
Convert the half-precision floating-point value h to an unsigned integer in round-to-nearest-even mode. NaN inputs are converted to 0.
__device__ unsigned int __half2uint_ru (const __half h)
Convert a half to an unsigned integer in round-up mode.

Parameters
h
- half. Is only being read.

Returns
unsigned int
- h converted to an unsigned integer.

Description
Convert the half-precision floating-point value h to an unsigned integer in round-up mode. NaN inputs are converted to 0.

__host__ __device__ unsigned int __half2uint_rz (const __half h)
Convert a half to an unsigned integer in round-towards-zero mode.

Parameters
h
- half. Is only being read.

Returns
unsigned int
- h converted to an unsigned integer.

Description
Convert the half-precision floating-point value h to an unsigned integer in round-towards-zero mode. NaN inputs are converted to 0.

__device__ unsigned long long int __half2ull_rd (const __half h)
Convert a half to an unsigned 64-bit integer in round-down mode.

Parameters
h
- half. Is only being read.
Returns
unsigned long long int

- \( h \) converted to an unsigned 64-bit integer.

Description
Convert the half-precision floating-point value \( h \) to an unsigned 64-bit integer in round-down mode. NaN inputs return 0x8000000000000000.

__device__ unsigned long long int __half2ull_rn (const __half h)
Convert a half to an unsigned 64-bit integer in round-to-nearest-even mode.

Parameters
h
- half. Is only being read.

Returns
unsigned long long int

- \( h \) converted to an unsigned 64-bit integer.

Description
Convert the half-precision floating-point value \( h \) to an unsigned 64-bit integer in round-to-nearest-even mode. NaN inputs return 0x8000000000000000.

__device__ unsigned long long int __half2ull_ru (const __half h)
Convert a half to an unsigned 64-bit integer in round-up mode.

Parameters
h
- half. Is only being read.

Returns
unsigned long long int

- \( h \) converted to an unsigned 64-bit integer.
Description
Convert the half-precision floating-point value $h$ to an unsigned 64-bit integer in round-up mode. NaN inputs return 0x8000000000000000.

`__host__ device__ unsigned long long int __half2ull_rz (const __half h)`
Convert a half to an unsigned 64-bit integer in round-towards-zero mode.

Parameters
- $h$ - half. Is only being read.

Returns
unsigned long long int
- $h$ converted to an unsigned 64-bit integer.

Description
Convert the half-precision floating-point value $h$ to an unsigned 64-bit integer in round-towards-zero mode. NaN inputs return 0x8000000000000000.

`__device__ unsigned short int __half2ushort_rd (const __half h)`
Convert a half to an unsigned short integer in round-down mode.

Parameters
- $h$ - half. Is only being read.

Returns
unsigned short int
- $h$ converted to an unsigned short integer.

Description
Convert the half-precision floating-point value $h$ to an unsigned short integer in round-down mode. NaN inputs are converted to 0.
__device__ unsigned short int __half2ushort_rn (const __half h)
Convert a half to an unsigned short integer in round-to-nearest-even mode.

Parameters
h
- half. Is only being read.

Returns
unsigned short int

 deprag. h converted to an unsigned short integer.

Description
Convert the half-precision floating-point value h to an unsigned short integer in round-to-nearest-even mode. NaN inputs are converted to 0.

__device__ unsigned short int __half2ushort_ru (const __half h)
Convert a half to an unsigned short integer in round-up mode.

Parameters
h
- half. Is only being read.

Returns
unsigned short int

 deprag. h converted to an unsigned short integer.

Description
Convert the half-precision floating-point value h to an unsigned short integer in round-up mode. NaN inputs are converted to 0.
__host__ __device__ unsigned short int __half2ushort_rz (const __half h)

Convert a half to an unsigned short integer in round-towards-zero mode.

Parameters

**h**
- half. Is only being read.

Returns

unsigned short int

- h converted to an unsigned short integer.

Description

Convert the half-precision floating-point value h to an unsigned short integer in round-towards-zero mode. NaN inputs are converted to 0.

__host__ __device__ short int __half_as_short (const __half h)

Reinterprets bits in a half as a signed short integer.

Parameters

**h**
- half. Is only being read.

Returns

short int

- The reinterpreted value.

Description

Reinterprets the bits in the half-precision floating-point number h as a signed short integer.
__host__ __device__ unsigned short int __half_as_ushort (const __half h)
Reinterprets bits in a half as an unsigned short integer.

Parameters

h
- half. Is only being read.

Returns

unsigned short int
➤ The reinterpreted value.

Description

Reinterprets the bits in the half-precision floating-point h as an unsigned short number.

__host__ __device__ halves2half2 (const __half a, const __half b)
Combines two half numbers into one half2 number.

Parameters

a
- half. Is only being read.
b
- half. Is only being read.

Returns

half2
➤ The half2 with one half equal to a and the other to b.

Description

Combines two input half number a and b into one half2 number. Input a is stored in low 16 bits of the return value, input b is stored in high 16 bits of the return value.
__host__ __device__ float __high2float (const __half2 a)
Converts high 16 bits of half2 to float and returns the result.

Parameters
a
- half2. Is only being read.

Returns
float
▶ The high 16 bits of a converted to float.

Description
Converts high 16 bits of half2 input a to 32-bit floating-point number and returns the result.

__host__ __device__ __high2half (const __half2 a)
Returns high 16 bits of half2 input.

Parameters
a
- half2. Is only being read.

Returns
half
▶ The high 16 bits of the input.

Description
Returns high 16 bits of half2 input a.

__host__ __device__ __high2half2 (const __half2 a)
Extracts high 16 bits from half2 input.

Parameters
a
- half2. Is only being read.

Returns
half2
▶ The half2 with both halves equal to the high 16 bits of the input.
Description
Extracts high 16 bits from \texttt{half2} input \texttt{a} and returns a new \texttt{half2} number which has both halves equal to the extracted bits.

\texttt{__host\_device\_highs2half2 (const \_half2 a, const \_half2 b)}
Extracts high 16 bits from each of the two \texttt{half2} inputs and combines into one \texttt{half2} number.

Parameters
\begin{itemize}
\item \texttt{a} - \texttt{half2}. Is only being read.
\item \texttt{b} - \texttt{half2}. Is only being read.
\end{itemize}

Returns
\begin{itemize}
\item \texttt{half2} - The high 16 bits of \texttt{a} and of \texttt{b}.
\end{itemize}

Description
Extracts high 16 bits from each of the two \texttt{half2} inputs and combines into one \texttt{half2} number. High 16 bits from input \texttt{a} is stored in low 16 bits of the return value, high 16 bits from input \texttt{b} is stored in high 16 bits of the return value.

\texttt{__host\_device\_int2half\_rd (const int i)}
Convert a signed integer to a half in round-down mode.

Parameters
\begin{itemize}
\item \texttt{i} - \texttt{int}. Is only being read.
\end{itemize}

Returns
\begin{itemize}
\item \texttt{half} - \texttt{i} converted to half.
\end{itemize}

Description
Convert the signed integer value \texttt{i} to a half-precision floating-point value in round-down mode.
__host__ __device__ int2half_rn (const int i)
Convert a signed integer to a half in round-to-nearest-even mode.

Parameters
i
- int. Is only being read.

Returns
half
  i converted to half.

Description
Convert the signed integer value i to a half-precision floating-point value in round-to-nearest-even mode.

__host__ __device__ int2half_ru (const int i)
Convert a signed integer to a half in round-up mode.

Parameters
i
- int. Is only being read.

Returns
half
  i converted to half.

Description
Convert the signed integer value i to a half-precision floating-point value in round-up mode.

__host__ __device__ int2half_rz (const int i)
Convert a signed integer to a half in round-towards-zero mode.

Parameters
i
- int. Is only being read.

Returns
half
- i converted to half.

**Description**
Convert the signed integer value i to a half-precision floating-point value in round-towards-zero mode.

```c
__device__ ldca (const __half *ptr)
```
Generates a `ld.global.ca` load instruction.

**Parameters**
- `ptr` - memory location

**Returns**
The value pointed by `ptr`.

```c
__device__ ldca (const __half2 *ptr)
```
Generates a `ld.global.ca` load instruction.

**Parameters**
- `ptr` - memory location

**Returns**
The value pointed by `ptr`.

```c
__device__ ldcg (const __half *ptr)
```
Generates a `ld.global.cg` load instruction.

**Parameters**
- `ptr` - memory location

**Returns**
The value pointed by `ptr`.
__device__ ldcg (const __half2 *ptr)
Generates a `ld.global.cg` load instruction.

Parameters
ptr
- memory location

Returns
The value pointed by `ptr`

__device__ ldcs (const __half *ptr)
Generates a `ld.global.cs` load instruction.

Parameters
ptr
- memory location

Returns
The value pointed by `ptr`

__device__ ldcs (const __half2 *ptr)
Generates a `ld.global.cs` load instruction.

Parameters
ptr
- memory location

Returns
The value pointed by `ptr`

__device__ ldcv (const __half *ptr)
Generates a `ld.global.cv` load instruction.

Parameters
ptr
- memory location

Returns
The value pointed by `ptr`
__device__ ldcv (const __half2 *ptr)
Generates a `ld.global.cv` load instruction.

Parameters
ptr  
- memory location

Returns
The value pointed by `ptr`

__device__ ldg (const __half *ptr)
Generates a `ld.global.nc` load instruction.

Parameters
ptr  
- memory location

Returns
The value pointed by `ptr`

__device__ ldg (const __half2 *ptr)
Generates a `ld.global.nc` load instruction.

Parameters
ptr  
- memory location

Returns
The value pointed by `ptr`

Description
defined(__CUDA_ARCH__) || (__CUDA_ARCH__ >= 300)

__device__ ldlu (const __half *ptr)
Generates a `ld.global.lu` load instruction.

Parameters
ptr  
- memory location
**__device____ldlu (const __half2 *ptr)**
Generates a `ld.global.lu` load instruction.

**Parameters**

**ptr**
- memory location

**Returns**
The value pointed by `ptr`

---

**__host______device____ll2half_rd (const long long int i)**
Convert a signed 64-bit integer to a half in round-down mode.

**Parameters**

**i**
- long long int. Is only being read.

**Returns**

half

➤ i converted to half.

**Description**
Convert the signed 64-bit integer value `i` to a half-precision floating-point value in round-down mode.

**__host______device____ll2half_rn (const long long int i)**
Convert a signed 64-bit integer to a half in round-to-nearest-even mode.

**Parameters**

**i**
- long long int. Is only being read.

**Returns**

half

➤ i converted to half.
Description
Convert the signed 64-bit integer value \(i\) to a half-precision floating-point value in round-to-nearest-even mode.

\[
\text{__host____device____ll2half_ru (const long long int} \ i)\]
Convert a signed 64-bit integer to a half in round-up mode.

Parameters
\(i\)
  - long long int. Is only being read.

Returns
half
  - \(i\) converted to half.

Description
Convert the signed 64-bit integer value \(i\) to a half-precision floating-point value in round-up mode.

\[
\text{__host____device____ll2half_rz (const long long int} \ i)\]
Convert a signed 64-bit integer to a half in round-towards-zero mode.

Parameters
\(i\)
  - long long int. Is only being read.

Returns
half
  - \(i\) converted to half.

Description
Convert the signed 64-bit integer value \(i\) to a half-precision floating-point value in round-towards-zero mode.
__host___device___ float __low2float (const __half2 a)
Converts low 16 bits of __half2 to float and returns the result.

Parameters
  a
  - __half2. Is only being read.

Returns
float
  - The low 16 bits of a converted to float.

Description
Converts low 16 bits of __half2 input a to 32-bit floating-point number and returns the result.

__host___device___low2half (const __half2 a)
Returns low 16 bits of __half2 input.

Parameters
  a
  - __half2. Is only being read.

Returns
half
  - Returns __half which contains low 16 bits of the input a.

Description
Returns low 16 bits of __half2 input a.

__host___device___low2half2 (const __half2 a)
Extracts low 16 bits from __half2 input.

Parameters
  a
  - __half2. Is only being read.

Returns
half2
  - The __half2 with both halves equal to the low 16 bits of the input.
Description
Extracts low 16 bits from \texttt{half2} input \texttt{a} and returns a new \texttt{half2} number which has both halves equal to the extracted bits.

\texttt{__host\_\_device\_\_\_\_lowhigh2highlow (const \_\_half2 a)}

Swaps both halves of the \texttt{half2} input.

Parameters
\texttt{a}
- \texttt{half2}. Is only being read.

Returns
\texttt{half2}
- \texttt{a} with its halves being swapped.

Description
Swaps both halves of the \texttt{half2} input and returns a new \texttt{half2} number with swapped halves.

\texttt{__host\_\_device\_\_\_\_lows2half2 (const \_\_half2 a, const \_\_half2 b)}

Extracts low 16 bits from each of the two \texttt{half2} inputs and combines into one \texttt{half2} number.

Parameters
\texttt{a}
- \texttt{half2}. Is only being read.
\texttt{b}
- \texttt{half2}. Is only being read.

Returns
\texttt{half2}
- The low 16 bits of \texttt{a} and of \texttt{b}.

Description
Extracts low 16 bits from each of the two \texttt{half2} inputs and combines into one \texttt{half2} number. Low 16 bits from input \texttt{a} is stored in low 16 bits of the return value, low 16 bits from input \texttt{b} is stored in high 16 bits of the return value.
__device__ __shfl_down_sync (const unsigned mask, const __half var, const unsigned int delta, const int width)

Exchange a variable between threads within a warp. Copy from a thread with higher ID relative to the caller.

**Parameters**

- `mask`: unsigned int. Is only being read.
- `var`: half. Is only being read.
- `delta`: int. Is only being read.
- `width`: int. Is only being read.

**Returns**

Returns the 2-byte word referenced by var from the source thread ID as half. If the source thread ID is out of range or the source thread has exited, the calling thread’s own var is returned.

**Description**

Calculates a source thread ID by adding delta to the caller’s thread ID. The value of var held by the resulting thread ID is returned: this has the effect of shifting var down the warp by delta threads. If width is less than warpSize then each subsection of the warp behaves as a separate entity with a starting logical thread ID of 0. As for __shfl_up_sync[], the ID number of the source thread will not wrap around the value of width and so the upper delta threads will remain unchanged.

**Note:**

For more details for this function see the Warp Shuffle Functions section in the CUDA C++ Programming Guide.
__device____shfl_down_sync (const unsigned mask, const __half2 var, const unsigned int delta, const int width)
Exchange a variable between threads within a warp. Copy from a thread with higher ID relative to the caller.

Parameters

mask
  - unsigned int. Is only being read.
var
  - half2. Is only being read.
delta
  - int. Is only being read.
width
  - int. Is only being read.

Returns
Returns the 4-byte word referenced by var from the source thread ID as half2. If the source thread ID is out of range or the source thread has exited, the calling thread’s own var is returned.

Description
Calculates a source thread ID by adding delta to the caller’s thread ID. The value of var held by the resulting thread ID is returned: this has the effect of shifting var down the warp by delta threads. If width is less than warpSize then each subsection of the warp behaves as a separate entity with a starting logical thread ID of 0. As for __shfl_up_sync[], the ID number of the source thread will not wrap around the value of width and so the upper delta threads will remain unchanged.

Note:
For more details for this function see the Warp Shuffle Functions section in the CUDA C++ Programming Guide.

__device____shfl_sync (const unsigned mask, const __half var, const int delta, const int width)
Exchange a variable between threads within a warp. Direct copy from indexed thread.

Parameters

mask
  - unsigned int. Is only being read.
var
  - half. Is only being read.
delta
  - int. Is only being read.
width
  - int. Is only being read.

Returns
Returns the 2-byte word referenced by var from the source thread ID as half. If the source thread ID is out of range or the source thread has exited, the calling thread’s own var is returned.

Description
Returns the value of var held by the thread whose ID is given by delta. If width is less than warpSize then each subsection of the warp behaves as a separate entity with a starting logical thread ID of 0. If delta is outside the range [0:width-1], the value returned corresponds to the value of var held by the delta modulo width (i.e. within the same subsection). width must have a value which is a power of 2; results are undefined if width is not a power of 2, or is a number greater than warpSize.

Note:
For more details for this function see the Warp Shuffle Functions section in the CUDA C++ Programming Guide.

__device__ shfl_sync (const unsigned mask, const __half2 var, const int delta, const int width)
Exchange a variable between threads within a warp. Direct copy from indexed thread.

Parameters

mask
  - unsigned int. Is only being read.
var
  - half2. Is only being read.
delta
  - int. Is only being read.
width
  - int. Is only being read.
Returns
Returns the 4-byte word referenced by var from the source thread ID as half2. If the source thread ID is out of range or the source thread has exited, the calling thread’s own var is returned.

Description
Returns the value of var held by the thread whose ID is given by delta. If width is less than warpSize then each subsection of the warp behaves as a separate entity with a starting logical thread ID of 0. If delta is outside the range [0:width-1], the value returned corresponds to the value of var held by the delta modulo width (i.e. within the same subsection). width must have a value which is a power of 2; results are undefined if width is not a power of 2, or is a number greater than warpSize.

`__device__ __shfl_up_sync (const unsigned mask, const __half var, const unsigned int delta, const int width)`
Exchange a variable between threads within a warp. Copy from a thread with lower ID relative to the caller.

Parameters
- **mask**
  - unsigned int. Is only being read.
- **var**
  - half. Is only being read.
- **delta**
  - int. Is only being read.
- **width**
  - int. Is only being read.

Returns
Returns the 2-byte word referenced by var from the source thread ID as half. If the source thread ID is out of range or the source thread has exited, the calling thread’s own var is returned.

Description
Calculates a source thread ID by subtracting delta from the caller’s lane ID. The value of var held by the resulting lane ID is returned: in effect, var is shifted up the warp by delta threads. If width is less than warpSize then each subsection of the warp behaves as a separate entity with a starting logical thread ID of 0. The source thread index will not wrap around the value of width, so effectively the lower delta threads will be unchanged. width must have a value which
is a power of 2; results are undefined if width is not a power of 2, or is a number greater than warpSize.

```
__device__ shfl_up_sync (const unsigned mask, const __half2 var, const unsigned int delta, const int width)
```

Exchange a variable between threads within a warp. Copy from a thread with lower ID relative to the caller.

**Parameters**

- **mask**  
  - unsigned int. Is only being read.
- **var**  
  - half2. Is only being read.
- **delta**  
  - int. Is only being read.
- **width**  
  - int. Is only being read.

**Returns**

Returns the 4-byte word referenced by var from the source thread ID as half2. If the source thread ID is out of range or the source thread has exited, the calling thread’s own var is returned.

**Description**

Calculates a source thread ID by subtracting delta from the caller’s lane ID. The value of var held by the resulting lane ID is returned: in effect, var is shifted up the warp by delta threads. If width is less than warpSize then each subsection of the warp behaves as a separate entity with a starting logical thread ID of 0. The source thread index will not wrap around the value of width, so effectively the lower delta threads will be unchanged. width must have a value which is a power of 2; results are undefined if width is not a power of 2, or is a number greater than warpSize.
```c
__device__ shfl_xor_sync (const unsigned mask, const __half var, const int delta, const int width)
```

Exchange a variable between threads within a warp. Copy from a thread based on bitwise XOR of own thread ID.

**Parameters**

- **mask**
  - unsigned int. Is only being read.
- **var**
  - half. Is only being read.
- **delta**
  - int. Is only being read.
- **width**
  - int. Is only being read.

**Returns**

Returns the 2-byte word referenced by var from the source thread ID as half. If the source thread ID is out of range or the source thread has exited, the calling thread’s own var is returned.

**Description**

Calculates a source thread ID by performing a bitwise XOR of the caller’s thread ID with mask: the value of var held by the resulting thread ID is returned. If width is less than warpSize then each group of width consecutive threads are able to access elements from earlier groups of threads, however if they attempt to access elements from later groups of threads their own value of var will be returned. This mode implements a butterfly addressing pattern such as is used in tree reduction and broadcast.

**Note:**

For more details for this function see the Warp Shuffle Functions section in the CUDA C++ Programming Guide.
__device__ shfl_xor_sync (const unsigned mask, const __half2 var, const int delta, const int width)

Exchange a variable between threads within a warp. Copy from a thread based on bitwise XOR of own thread ID.

**Parameters**

- **mask**
  - unsigned int. Is only being read.
- **var**
  - half2. Is only being read.
- **delta**
  - int. Is only being read.
- **width**
  - int. Is only being read.

**Returns**

Returns the 4-byte word referenced by var from the source thread ID as half2. If the source thread ID is out of range or the source thread has exited, the calling thread’s own var is returned.

**Description**

Calculates a source thread ID by performing a bitwise XOR of the caller’s thread ID with mask: the value of var held by the resulting thread ID is returned. If width is less than warpSize then each group of width consecutive threads are able to access elements from earlier groups of threads, however if they attempt to access elements from later groups of threads their own value of var will be returned. This mode implements a butterfly addressing pattern such as is used in tree reduction and broadcast.

**Note:**

For more details for this function see the Warp Shuffle Functions section in the CUDA C++ Programming Guide.

__host__ __device__ short2half_rd (const short int i)

Convert a signed short integer to a half in round-down mode.

**Parameters**

- **i**
  - short int. Is only being read.
>Returns
half
- i converted to half.

_Description_
Convert the signed short integer value i to a half-precision floating-point value in round-down mode.

`__host__device__short2half_rn (const short int i)`
Convert a signed short integer to a half in round-to-nearest-even mode.

_Parameters_
i
- short int. Is only being read.

_Returns_
half
- i converted to half.

_Description_
Convert the signed short integer value i to a half-precision floating-point value in round-to-nearest-even mode.

`__host__device__short2half_ru (const short int i)`
Convert a signed short integer to a half in round-up mode.

_Parameters_
i
- short int. Is only being read.

_Returns_
half
- i converted to half.

_Description_
Convert the signed short integer value i to a half-precision floating-point value in round-up mode.
__host__ __device__ short2half_rz (const short int i)
Convert a signed short integer to a half in round-towards-zero mode.

Parameters
i
- short int. Is only being read.

Returns
half
- i converted to half.

Description
Convert the signed short integer value \( i \) to a half-precision floating-point value in round-towards-zero mode.

__host__ __device__ short_as_half (const short int i)
Reinterprets bits in a signed short integer as a half.

Parameters
i
- short int. Is only being read.

Returns
half
- The reinterpreted value.

Description
Reinterprets the bits in the signed short integer \( i \) as a half-precision floating-point number.

__device__ void __stcg (const __half *ptr, const __half value)
Generates a `st.global.cg` store instruction.

Parameters
ptr
- memory location
value
- the value to be stored
__device__ void __stcg (const __half2 *ptr, const __half2 value)
Generates a `st.global.cg` store instruction.

Parameters
ptr
- memory location
value
- the value to be stored

__device__ void __stcs (const __half *ptr, const __half value)
Generates a `st.global.cs` store instruction.

Parameters
ptr
- memory location
value
- the value to be stored

__device__ void __stcs (const __half2 *ptr, const __half2 value)
Generates a `st.global.cs` store instruction.

Parameters
ptr
- memory location
value
- the value to be stored

__device__ void __stwb (const __half *ptr, const __half value)
Generates a `st.global.wb` store instruction.

Parameters
ptr
- memory location
value
- the value to be stored
__device__ void __stwb (const __half2 *ptr, const __half2 value)
Generates a `st.global.wb` store instruction.

Parameters
ptr
- memory location
value
- the value to be stored

__device__ void __stwt (const __half *ptr, const __half value)
Generates a `st.global.wt` store instruction.

Parameters
ptr
- memory location
value
- the value to be stored

__device__ void __stwt (const __half2 *ptr, const __half2 value)
Generates a `st.global.wt` store instruction.

Parameters
ptr
- memory location
value
- the value to be stored

__host____device____uint2half_rd (const unsigned int i)
Convert an unsigned integer to a half in round-down mode.

Parameters
i
- unsigned int. Is only being read.

Returns
half
i converted to half.

**Description**
Convert the unsigned integer value i to a half-precision floating-point value in round-down mode.

```c
__host__ __device__ uint2half_rn (const unsigned int i)
```

**Parameters**
- i - unsigned int. Is only being read.

**Returns**
- half
  - i converted to half.

**Description**
Convert the unsigned integer value i to a half-precision floating-point value in round-to-nearest-even mode.

```c
__host__ __device__ uint2half_ru (const unsigned int i)
```

**Parameters**
- i - unsigned int. Is only being read.

**Returns**
- half
  - i converted to half.

**Description**
Convert the unsigned integer value i to a half-precision floating-point value in round-up mode.
__host__ __device__ uint2half_rz (const unsigned int i)
Convert an unsigned integer to a half in round-towards-zero mode.

Parameters
i
  - unsigned int. Is only being read.

Returns
half
  i converted to half.

Description
Convert the unsigned integer value i to a half-precision floating-point value in round-towards-zero mode.

__host__ __device__ ull2half_rd (const unsigned long long int i)
Convert an unsigned 64-bit integer to a half in round-down mode.

Parameters
i
  - unsigned long long int. Is only being read.

Returns
half
  i converted to half.

Description
Convert the unsigned 64-bit integer value i to a half-precision floating-point value in round-down mode.

__host__ __device__ ull2half_rn (const unsigned long long int i)
Convert an unsigned 64-bit integer to a half in round-to-nearest-even mode.

Parameters
i
  - unsigned long long int. Is only being read.
Returns
half
► i converted to half.

Description
Convert the unsigned 64-bit integer value i to a half-precision floating-point value in round-to-nearest-even mode.

__host__ __device__ ull2half_ru (const unsigned long long int i)
Convert an unsigned 64-bit integer to a half in round-up mode.

Parameters
i
- unsigned long long int. Is only being read.

Returns
half
► i converted to half.

Description
Convert the unsigned 64-bit integer value i to a half-precision floating-point value in round-up mode.

__host__ __device__ ull2half_rz (const unsigned long long int i)
Convert an unsigned 64-bit integer to a half in round-towards-zero mode.

Parameters
i
- unsigned long long int. Is only being read.

Returns
half
► i converted to half.
Description
Convert the unsigned 64-bit integer value \( i \) to a half-precision floating-point value in round-towards-zero mode.

__host____device____ushort2half_rd (const unsigned short int \( i \))
Convert an unsigned short integer to a half in round-down mode.

Parameters
\( i \)
- unsigned short int. Is only being read.

Returns
half
- \( i \) converted to half.

Description
Convert the unsigned short integer value \( i \) to a half-precision floating-point value in round-down mode.

__host____device____ushort2half_rn (const unsigned short int \( i \))
Convert an unsigned short integer to a half in round-to-nearest-even mode.

Parameters
\( i \)
- unsigned short int. Is only being read.

Returns
half
- \( i \) converted to half.

Description
Convert the unsigned short integer value \( i \) to a half-precision floating-point value in round-to-nearest-even mode.
__host__ __device__ ushort2half_ru (const unsigned short int i)
Convert an unsigned short integer to a half in round-up mode.

Parameters
i
- unsigned short int. Is only being read.

Returns
half
- i converted to half.

Description
Convert the unsigned short integer value i to a half-precision floating-point value in round-up mode.

__host__ __device__ ushort2half_rz (const unsigned short int i)
Convert an unsigned short integer to a half in round-towards-zero mode.

Parameters
i
- unsigned short int. Is only being read.

Returns
half
- i converted to half.

Description
Convert the unsigned short integer value i to a half-precision floating-point value in round-towards-zero mode.
`__host__ __device__ ushort_as_half (const unsigned short int i)`
Reinterprets bits in an unsigned short integer as a `half`.

**Parameters**

i
- unsigned short int. Is only being read.

**Returns**

half

- The reinterpreted value.

**Description**
Reinterprets the bits in the unsigned short integer i as a half-precision floating-point number.

`__host__ __device__ make_half2 (const __half x, const __half y)`
Vector function, combines two __half numbers into one __half2 number.

**Parameters**

x
- half. Is only being read.

y
- half. Is only being read.

**Returns**

__half2

- The __half2 vector with one half equal to x and the other to y.

**Description**
Combines two input __half number x and y into one __half2 number. Input x is stored in low 16 bits of the return value, input y is stored in high 16 bits of the return value.

### 1.2.7. Half Math Functions

**Half Precision Intrinsics**

To use these functions, include the header file `cuda_fp16.h` in your program.
\_\_device\_\_hcei1 (const \_\_half h)
Calculate ceiling of the input argument.

Parameters

\( \text{h} \)
- half. Is only being read.

Returns

half
- The smallest integer value not less than \( h \).

Description

Compute the smallest integer value not less than \( h \).

\_\_device\_\_hc0s (const \_\_half a)
Calculates \( \text{half} \) cosine in round-to-nearest-even mode.

Parameters

\( \text{a} \)
- half. Is only being read.

Returns

half
- The cosine of \( a \).

Description

Calculates \( \text{half} \) cosine of input \( a \) in round-to-nearest-even mode.

\_\_device\_\_hexp (const \_\_half a)
Calculates \( \text{half} \) natural exponential function in round-to-nearest-even mode.

Parameters

\( \text{a} \)
- half. Is only being read.

Returns

half
- The natural exponential function on \( a \).
Description
Calculates half natural exponential function of input a in round-to-nearest-even mode.

__device__hexp10 (const __half a)
Calculates half decimal exponential function in round-to-nearest-even mode.

Parameters
a
  - half. Is only being read.

Returns
half
  ▶ The decimal exponential function on a.

Description
Calculates half decimal exponential function of input a in round-to-nearest-even mode.

__device__hexp2 (const __half a)
Calculates half binary exponential function in round-to-nearest-even mode.

Parameters
a
  - half. Is only being read.

Returns
half
  ▶ The binary exponential function on a.

Description
Calculates half binary exponential function of input a in round-to-nearest-even mode.

__device__hfloor (const __half h)
Calculate the largest integer less than or equal to h.

Parameters
h
  - half. Is only being read.
Returns
half
- The largest integer value which is less than or equal to h.

Description
Calculate the largest integer value which is less than or equal to h.

__device__hlog (const __half a)
Calculates half natural logarithm in round-to-nearest-even mode.

Parameters
a
- half. Is only being read.

Returns
half
- The natural logarithm of a.

Description
Calculates half natural logarithm of input a in round-to-nearest-even mode.

__device__hlog10 (const __half a)
Calculates half decimal logarithm in round-to-nearest-even mode.

Parameters
a
- half. Is only being read.

Returns
half
- The decimal logarithm of a.

Description
Calculates half decimal logarithm of input a in round-to-nearest-even mode.
__device__ hlog2 (const __half a)
Calculates half binary logarithm in round-to-nearest-even mode.

Parameters
a
- half. Is only being read.

Returns
half
- The binary logarithm of a.

Description
Calculates half binary logarithm of input a in round-to-nearest-even mode.

__device__ hrcp (const __half a)
Calculates half reciprocal in round-to-nearest-even mode.

Parameters
a
- half. Is only being read.

Returns
half
- The reciprocal of a.

Description
Calculates half reciprocal of input a in round-to-nearest-even mode.

__device__ hrint (const __half h)
Round input to nearest integer value in half-precision floating-point number.

Parameters
h
- half. Is only being read.

Returns
half
- The nearest integer to h.
Description
Round \( h \) to the nearest integer value in half-precision floating-point format, with halfway cases rounded to the nearest even integer value.

__device__hrsqrt (const __half a)
Calculates \( \text{half} \) reciprocal square root in round-to-nearest-even mode.

Parameters
a
- half. Is only being read.

Returns
half
- The reciprocal square root of \( a \).

Description
Calculates \( \text{half} \) reciprocal square root of input \( a \) in round-to-nearest-even mode.

__device__hsin (const __half a)
Calculates \( \text{half} \) sine in round-to-nearest-even mode.

Parameters
a
- half. Is only being read.

Returns
half
- The sine of \( a \).

Description
Calculates \( \text{half} \) sine of input \( a \) in round-to-nearest-even mode.
__device__ hsqrt (const __half a)
Calculates half square root in round-to-nearest-even mode.

Parameters
a
  - half. Is only being read.

Returns
half
  ▶ The square root of a.

Description
Calculates half square root of input a in round-to-nearest-even mode.

__device__ htrunc (const __half h)
Truncate input argument to the integral part.

Parameters
h
  - half. Is only being read.

Returns
half
  ▶ The truncated integer value.

Description
Round h to the nearest integer value that does not exceed h in magnitude.

1.2.8. Half2 Math Functions
Half Precision Intrinsics
To use these functions, include the header file cuda_fp16.h in your program.


__device__h2ceil (const __half2 h)
Calculate half2 vector ceiling of the input argument.

**Parameters**

**h**
- half2. Is only being read.

**Returns**

half2
- The vector of smallest integers not less than h.

**Description**

For each component of vector h compute the smallest integer value not less than h.

__device__h2cos (const __half2 a)
Calculates half2 vector cosine in round-to-nearest-even mode.

**Parameters**

**a**
- half2. Is only being read.

**Returns**

half2
- The elementwise cosine on vector a.

**Description**

Calculates half2 cosine of input vector a in round-to-nearest-even mode.

__device__h2exp (const __half2 a)
Calculates half2 vector exponential function in round-to-nearest-even mode.

**Parameters**

**a**
- half2. Is only being read.

**Returns**

half2
- The elementwise exponential function on vector a.
Description
Calculates $\text{half2}$ exponential function of input vector $a$ in round-to-nearest-even mode.

$__device__\text{h2exp10} (\text{const } \text{half2} \ a)\$
Calculates $\text{half2}$ vector decimal exponential function in round-to-nearest-even mode.

Parameters
\begin{itemize}
  \item \text{a}
  \begin{itemize}
    \item - $\text{half2}$. Is only being read.
  \end{itemize}
\end{itemize}

Returns
$\text{half2}$
\begin{itemize}
  \item The elementwise decimal exponential function on vector $a$.
\end{itemize}

Description
Calculates $\text{half2}$ decimal exponential function of input vector $a$ in round-to-nearest-even mode.

$__device__\text{h2exp2} (\text{const } \text{half2} \ a)\$
Calculates $\text{half2}$ vector binary exponential function in round-to-nearest-even mode.

Parameters
\begin{itemize}
  \item \text{a}
  \begin{itemize}
    \item - $\text{half2}$. Is only being read.
  \end{itemize}
\end{itemize}

Returns
$\text{half2}$
\begin{itemize}
  \item The elementwise binary exponential function on vector $a$.
\end{itemize}

Description
Calculates $\text{half2}$ binary exponential function of input vector $a$ in round-to-nearest-even mode.
\texttt{__device\_h2floor (const \_half2 h)}

Calculate the largest integer less than or equal to \( h \).

\textbf{Parameters}

\( h \)

- \_half2. Is only being read.

\textbf{Returns}

\_half2

- The vector of largest integers which is less than or equal to \( h \).

\textbf{Description}

For each component of vector \( h \) calculate the largest integer value which is less than or equal to \( h \).

\texttt{__device\_h2log (const \_half2 a)}

Calculates \_half2 vector natural logarithm in round-to-nearest-even mode.

\textbf{Parameters}

\( a \)

- \_half2. Is only being read.

\textbf{Returns}

\_half2

- The elementwise natural logarithm on vector \( a \).

\textbf{Description}

Calculates \_half2 natural logarithm of input vector \( a \) in round-to-nearest-even mode.

\texttt{__device\_h2log10 (const \_half2 a)}

Calculates \_half2 vector decimal logarithm in round-to-nearest-even mode.

\textbf{Parameters}

\( a \)

- \_half2. Is only being read.

\textbf{Returns}

\_half2
The elementwise decimal logarithm on vector a.

Description
Calculates half2 decimal logarithm of input vector a in round-to-nearest-even mode.

__device__h2log2 (const __half2 a)
Calculates half2 vector binary logarithm in round-to-nearest-even mode.

Parameters
a
- half2. Is only being read.

Returns
half2
- The elementwise binary logarithm on vector a.

Description
Calculates half2 binary logarithm of input vector a in round-to-nearest-even mode.

__device__h2rcp (const __half2 a)
Calculates half2 vector reciprocal in round-to-nearest-even mode.

Parameters
a
- half2. Is only being read.

Returns
half2
- The elementwise reciprocal on vector a.

Description
Calculates half2 reciprocal of input vector a in round-to-nearest-even mode.
__device__h2rint (const __half2 h)
Round input to nearest integer value in half-precision floating-point number.

Parameters
h
- half2. Is only being read.

Returns
half2
- The vector of rounded integer values.

Description
Round each component of half2 vector h to the nearest integer value in half-precision floating-point format, with halfway cases rounded to the nearest even integer value.

__device__h2rsqrt (const __half2 a)
Calculates half2 vector reciprocal square root in round-to-nearest-even mode.

Parameters
a
- half2. Is only being read.

Returns
half2
- The elementwise reciprocal square root on vector a.

Description
Calculates half2 reciprocal square root of input vector a in round-to-nearest-even mode.

__device__h2sin (const __half2 a)
Calculates half2 vector sine in round-to-nearest-even mode.

Parameters
a
- half2. Is only being read.

Returns
half2
1.3. Bfloat16 Precision Intrinsics

This section describes nv_bfloat16 precision intrinsic functions. To use these functions, include the header file cuda_bf16.h in your program. All of the functions defined here are
available in device code. Some of the functions are also available to host compilers, please refer to respective functions’ documentation for details.

NOTE: Aggressive floating-point optimizations performed by host or device compilers may affect numeric behavior of the functions implemented in this header. Specific examples are:

- `hsin(__nv_bfloat16);`
- `hcos(__nv_bfloat16);`
- `h2sin(__nv_bfloat162);`
- `h2cos(__nv_bfloat162);`

The following macros are available to help users selectively enable/disable various definitions present in the header file:

- `CUDA_NO_BFLOAT16` - If defined, this macro will prevent the definition of additional type aliases in the global namespace, helping to avoid potential conflicts with symbols defined in the user program.

- `__CUDA_NO_BFLOAT16_CONVERSIONS__` - If defined, this macro will prevent the use of the C++ type conversions (converting constructors and conversion operators) that are common for built-in floating-point types, but may be undesirable for `__nv_bfloat16` which is essentially a user-defined type.

- `__CUDA_NO_BFLOAT16_OPERATORS__` and `__CUDA_NO_BFLOAT162_OPERATORS__` - If defined, these macros will prevent the inadvertent use of usual arithmetic and comparison operators. This enforces the storage-only type semantics and prevents C++ style computations on `__nv_bfloat16` and `__nv_bfloat162` types.

```c
struct __nv_bfloat16
nv_bfloat16 datatype

struct __nv_bfloat162
nv_bfloat162 datatype

struct __nv_bfloat162_raw
__nv_bfloat162_raw data type

struct __nv_bfloat16_raw
__nv_bfloat16_raw data type
```

Bfloat16 Arithmetic Constants

Bfloat16 Arithmetic Functions
Bfloat162 Arithmetic Functions
Bfloat16 Comparison Functions
Bfloat162 Comparison Functions
Bfloat16 Precision Conversion and Data Movement
Bfloat16 Math Functions
Bfloat162 Math Functions

typedef nv_bfloat16
This datatype is meant to be the first-class or fundamental implementation of the bfloat16 numbers format.
Should be implemented in the compiler in the future. Current implementation is a simple typedef to a respective user-level type with underscores.

typedef nv_bfloat162
This datatype is meant to be the first-class or fundamental implementation of type for pairs of bfloat16 numbers.
Should be implemented in the compiler in the future. Current implementation is a simple typedef to a respective user-level type with underscores.

1.3.1. Bfloat16 Arithmetic Constants
Bfloat16 Precision Intrinsics
To use these constants, include the header file cuda_bf16.h in your program.

#define CUDART_INF_BF16
__ushort_as_bfloat16((__unsigned short)0x7F80U)
Defines floating-point positive infinity value for the nv_bfloat16 data type.
#define CUDART_MAX_NORMAL_BF16
__ushort_as_bfloat16((unsigned short)0x7F7FU)
Defines a maximum representable value for the nv_bfloat16 data type.

#define CUDART_MIN_DENORM_BF16
__ushort_as_bfloat16((unsigned short)0x0001U)
Defines a minimum representable (denormalized) value for the nv_bfloat16 data type.

#define CUDART_NAN_BF16
__ushort_as_bfloat16((unsigned short)0x7FFFU)
Defines canonical NaN value for the nv_bfloat16 data type.

#define CUDART_NEG_ZERO_BF16
__ushort_as_bfloat16((unsigned short)0x8000U)
Defines a negative zero value for the nv_bfloat16 data type.

#define CUDART_ONE_BF16
__ushort_as_bfloat16((unsigned short)0x3F80U)
Defines a value of 1.0 for the nv_bfloat16 data type.

#define CUDART_ZERO_BF16
__ushort_as_bfloat16((unsigned short)0x0000U)
Defines a positive zero value for the nv_bfloat16 data type.

1.3.2. Bfloat16 Arithmetic Functions

Bfloat16 Precision Intrinsics

To use these functions, include the header file cuda_bf16.h in your program.

__host_____device_____h2div (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs nv_bfloat162 vector division in round-to-nearest-even mode.

Description

Divides nv_bfloat162 input vector a by input vector b in round-to-nearest-even mode.
__host__ __device__ habs (const __nv_bfloat16 a)
Calculates the absolute value of input nv_bfloat16 number and returns the result.

Parameters
a
- nv_bfloat16. Is only being read.

Returns
nv_bfloat16
- The absolute value of a.

Description
Calculates the absolute value of input nv_bfloat16 number and returns the result.

__host__ __device__ hadd (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs nv_bfloat16 addition in round-to-nearest-even mode.

Description
Performs nv_bfloat16 addition of inputs a and b, in round-to-nearest-even mode.

__host__ __device__ hadd_rn (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs nv_bfloat16 addition in round-to-nearest-even mode.

Description
Performs nv_bfloat16 addition of inputs a and b, in round-to-nearest-even mode. Prevents floating-point contractions of mul+add into fma.

__host__ __device__ hadd_sat (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs nv_bfloat16 addition in round-to-nearest-even mode, with saturation to [0.0, 1.0].

Parameters
a
- nv_bfloat16. Is only being read.
b
- nv_bfloat16. Is only being read.
Returns

nv_bfloat16

- The sum of \(a\) and \(b\), with respect to saturation.

Description

Performs \(\text{nv\_bfloat16}\) add of inputs \(a\) and \(b\), in round-to-nearest-even mode, and clamps the result to range [0.0, 1.0]. NaN results are flushed to +0.0.

```c
__host__ __device__ hdiv (const __nv_bfloat16 a, const __nv_bfloat16 b)
```

Performs \(\text{nv\_bfloat16}\) division in round-to-nearest-even mode.

Description

Divides \(\text{nv\_bfloat16}\) input \(a\) by input \(b\) in round-to-nearest-even mode.

```c
__device__ hfma (const __nv_bfloat16 a, const __nv_bfloat16 b, const __nv_bfloat16 c)
```

Performs \(\text{nv\_bfloat16}\) fused multiply-add in round-to-nearest-even mode.

Description

Performs \(\text{nv\_bfloat16}\) multiply on inputs \(a\) and \(b\), then performs a \(\text{nv\_bfloat16}\) add of the result with \(c\), rounding the result once in round-to-nearest-even mode.

```c
__device__ hfma_relu (const __nv_bfloat16 a, const __nv_bfloat16 b, const __nv_bfloat16 c)
```

Performs \(\text{nv\_bfloat16}\) fused multiply-add in round-to-nearest-even mode with relu saturation.

Parameters

- \(a\) - \(\text{nv\_bfloat16}\). Is only being read.
- \(b\) - \(\text{nv\_bfloat16}\). Is only being read.
- \(c\) - \(\text{nv\_bfloat16}\). Is only being read.

Returns

nv_bfloat16
The result of fused multiply-add operation on \(a\), \(b\), and \(c\) with relu saturation.

**Description**

Performs \texttt{nv\_bf\_float16} multiply on inputs \(a\) and \(b\), then performs a \texttt{nv\_bf\_float16} add of the result with \(c\), rounding the result once in round-to-nearest-even mode. Then negative result is clamped to 0. NaN result is converted to canonical NaN.

\[
\text{\_device\_hfma\_sat (const \_nv\_bf\_float16 a, const \_nv\_bf\_float16 b, const \_nv\_bf\_float16 c)}
\]

Performs \texttt{nv\_bf\_float16} fused multiply-add in round-to-nearest-even mode, with saturation to \([0.0, 1.0]\).

**Parameters**

- \(a\) - \texttt{nv\_bf\_float16}. Is only being read.
- \(b\) - \texttt{nv\_bf\_float16}. Is only being read.
- \(c\) - \texttt{nv\_bf\_float16}. Is only being read.

**Returns**

\texttt{nv\_bf\_float16}

The result of fused multiply-add operation on \(a\), \(b\), and \(c\), with respect to saturation.

**Description**

Performs \texttt{nv\_bf\_float16} multiply on inputs \(a\) and \(b\), then performs a \texttt{nv\_bf\_float16} add of the result with \(c\), rounding the result once in round-to-nearest-even mode, and clamps the result to range \([0.0, 1.0]\). NaN results are flushed to +0.0.

\[
\text{\_host\_device\_hmul (const \_nv\_bf\_float16 a, const \_nv\_bf\_float16 b)}
\]

Performs \texttt{nv\_bf\_float16} multiplication in round-to-nearest-even mode.

**Description**

Performs \texttt{nv\_bf\_float16} multiplication of inputs \(a\) and \(b\), in round-to-nearest-even mode.
__host__ __device__ hmul_rn (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs \texttt{nv}_\texttt{bfloat16} multiplication in round-to-nearest-even mode.

**Description**
Performs \texttt{nv}_\texttt{bfloat16} multiplication of inputs \(a\) and \(b\), in round-to-nearest-even mode. Prevents floating-point contractions of \texttt{mul+add} or \texttt{sub} into \texttt{fma}.

__host__ __device__ hmul_sat (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs \texttt{nv}_\texttt{bfloat16} multiplication in round-to-nearest-even mode, with saturation to \([0.0, 1.0]\).

**Parameters**
a - \texttt{nv}_\texttt{bfloat16}. Is only being read.
b - \texttt{nv}_\texttt{bfloat16}. Is only being read.

**Returns**
\texttt{nv}_\texttt{bfloat16}
- The result of multiplying \(a\) and \(b\), with respect to saturation.

**Description**
Performs \texttt{nv}_\texttt{bfloat16} multiplication of inputs \(a\) and \(b\), in round-to-nearest-even mode, and clamps the result to range \([0.0, 1.0]\). NaN results are flushed to +0.0.

__host__ __device__ hneg (const __nv_bfloat16 a)
Negates input \texttt{nv}_\texttt{bfloat16} number and returns the result.

**Description**
Negates input \texttt{nv}_\texttt{bfloat16} number and returns the result.
__host__ __device__ hsub (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs \texttt{nv\_bfloat16} subtraction in round-to-nearest-even mode.

Description
Subtracts \texttt{nv\_bfloat16} input \( b \) from input \( a \) in round-to-nearest-even mode.

__host__ __device__ hsub\_rn (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs \texttt{nv\_bfloat16} subtraction in round-to-nearest-even mode.

Description
Subtracts \texttt{nv\_bfloat16} input \( b \) from input \( a \) in round-to-nearest-even mode. Prevents floating-point contractions of mul+sub into fma.

__host__ __device__ hsub\_sat (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs \texttt{nv\_bfloat16} subtraction in round-to-nearest-even mode, with saturation to \([0.0, 1.0]\).

Parameters
\( a \)
- \texttt{nv\_bfloat16}. Is only being read.

\( b \)
- \texttt{nv\_bfloat16}. Is only being read.

Returns
\texttt{nv\_bfloat16}
- The result of subtraction of \( b \) from \( a \), with respect to saturation.

Description
Subtracts \texttt{nv\_bfloat16} input \( b \) from input \( a \) in round-to-nearest-even mode, and clamps the result to range \([0.0, 1.0]\). NaN results are flushed to +0.0.
**__device__atomicAdd (const __nv_bfloat16 *address, const __nv_bfloat16 val)**

Adds `val` to the value stored at `address` in global or shared memory, and writes this value back to `address`. This operation is performed in one atomic operation.

**Parameters**

- **address**
  - `__nv_bfloat16*`. An address in global or shared memory.
- **val**
  - `__nv_bfloat16`. The value to be added.

**Returns**

- `__nv_bfloat16`

  - The old value read from `address`.

**Description**

The location of `address` must be in global or shared memory. This operation has undefined behavior otherwise. This operation is natively supported by devices of compute capability 9.x and higher, older devices of compute capability 7.x and 8.x use emulation path.

---

**Note:**

For more details for this function see the Atomic Functions section in the CUDA C++ Programming Guide.

---

**__host__device__operator* (const __nv_bfloat16 lh, const __nv_bfloat16 rh)**

**Description**

Performs `nv_bfloat16` multiplication operation. See also `__hmul(__nv_bfloat16, __nv_bfloat16)`.

**__host__device__operator*= (__nv_bfloat16 lh, const __nv_bfloat16 rh)**

**Description**

Performs `nv_bfloat16` compound assignment with multiplication operation.
__host__ __device__ operator+ (const __nv_bfloat16 h)

Description
Implements \(\text{nv\_bfloat16}\) unary plus operator, returns input value.

__host__ __device__ operator+ (const __nv_bfloat16 lh, const __nv_bfloat16 rh)

Description
Performs \(\text{nv\_bfloat16}\) addition operation. See also __hadd( __nv_bfloat16, __nv_bfloat16)

__host__ __device__ operator++ (__nv_bfloat16 h, const int ignored)

Description
Performs \(\text{nv\_bfloat16}\) postfix increment operation.

__host__ __device__ operator++ (__nv_bfloat16 h)

Description
Performs \(\text{nv\_bfloat16}\) prefix increment operation.

__host__ __device__ operator+= (__nv_bfloat16 lh, const __nv_bfloat16 rh)

Description
Performs \(\text{nv\_bfloat16}\) compound assignment with addition operation.

__host__ __device__ operator- (const __nv_bfloat16 h)

Description
Implements \(\text{nv\_bfloat16}\) unary minus operator. See also __hneg( __nv_bfloat16)
__host__ __device__ operator- (const __nv_bfloat16 lh, const __nv_bfloat16 rh)

Description
Performs __nv_bfloat16 subtraction operation. See also __hsub(__nv_bfloat16, __nv_bfloat16)

__host__ __device__ operator-- (__nv_bfloat16 h, const int ignored)

Description
Performs __nv_bfloat16 postfix decrement operation.

__host__ __device__ operator-- (__nv_bfloat16 h)

Description
Performs __nv_bfloat16 prefix decrement operation.

__host__ __device__ operator-= (__nv_bfloat16 lh, const __nv_bfloat16 rh)

Description
Performs __nv_bfloat16 compound assignment with subtraction operation.

__host__ __device__ operator/ (const __nv_bfloat16 lh, const __nv_bfloat16 rh)

Description
Performs __nv_bfloat16 division operation. See also __hdiv(__nv_bfloat16, __nv_bfloat16)

__host__ __device__ operator/= (__nv_bfloat16 lh, const __nv_bfloat16 rh)

Description
Performs __nv_bfloat16 compound assignment with division operation.
1.3.3. **Bfloat162 Arithmetic Functions**

**Bfloat16 Precision Intrinsics**

To use these functions, include the header file `cuda_bf16.h` in your program.

```c
__host__ __device__ habs2 (const __nv_bfloat162 a)
```

Calculates the absolute value of both halves of the input `nv_bfloat162` number and returns the result.

**Parameters**

- `a` - `nv_bfloat162`. Is only being read.

**Returns**

- `bfloat2`
  - Returns `a` with the absolute value of both halves.

**Description**

Calculates the absolute value of both halves of the input `nv_bfloat162` number and returns the result.

```c
__host__ __device__ hadd2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
```

Performs `nv_bfloat162` vector addition in round-to-nearest-even mode.

**Description**

Performs `nv_bfloat162` vector add of inputs `a` and `b`, in round-to-nearest-even mode.

```c
__host__ __device__ hadd2_rn (const __nv_bfloat162 a, const __nv_bfloat162 b)
```

Performs `nv_bfloat162` vector addition in round-to-nearest-even mode.

**Description**

Performs `nv_bfloat162` vector add of inputs `a` and `b`, in round-to-nearest-even mode. Prevents floating-point contractions of `mul+add` into `fma`. 
__host__ __device__ hadd2_sat (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs nv_bfloat162 vector addition in round-to-nearest-even mode, with saturation to [0.0, 1.0].

Parameters

a
- nv_bfloat162. Is only being read.

b
- nv_bfloat162. Is only being read.

Returns

nv_bfloat162

- The sum of a and b, with respect to saturation.

Description

Performs nv_bfloat162 vector add of inputs a and b, in round-to-nearest-even mode, and clamps the results to range [0.0, 1.0]. NaN results are flushed to +0.0.

__device__ hcmadd (const __nv_bfloat162 a, const __nv_bfloat162 b, const __nv_bfloat162 c)
Performs fast complex multiply-accumulate.

Parameters

a
- nv_bfloat162. Is only being read.

b
- nv_bfloat162. Is only being read.

c
- nv_bfloat162. Is only being read.

Returns

nv_bfloat162

- The result of complex multiply-accumulate operation on complex numbers a, b, and c

Description

Interprets vector nv_bfloat162 input pairs a, b, and c as complex numbers in nv_bfloat16 precision and performs complex multiply-accumulate operation: a*b + c
__device__ __hfma2 (const __nv_bfloat162 a, const __nv_bfloat162 b, const __nv_bfloat162 c)
Performs __nv_bfloat162 vector fused multiply-add in round-to-nearest-even mode.

**Description**
Performs __nv_bfloat162 vector multiply on inputs a and b, then performs a __nv_bfloat162 vector add of the result with c, rounding the result once in round-to-nearest-even mode.

__device__ __hfma2_relu (const __nv_bfloat162 a, const __nv_bfloat162 b, const __nv_bfloat162 c)
Performs __nv_bfloat162 vector fused multiply-add in round-to-nearest-even mode with relu saturation.

**Parameters**
- **a** - __nv_bfloat162. Is only being read.
- **b** - __nv_bfloat162. Is only being read.
- **c** - __nv_bfloat162. Is only being read.

**Returns**
__nv_bfloat162

- The result of elementwise fused multiply-add operation on vectors a, b, and c with relu saturation.

**Description**
Performs __nv_bfloat162 vector multiply on inputs a and b, then performs a __nv_bfloat162 vector add of the result with c, rounding the result once in round-to-nearest-even mode. Then negative result is clamped to 0. NaN result is converted to canonical NaN.

__device__ __hfma2_sat (const __nv_bfloat162 a, const __nv_bfloat162 b, const __nv_bfloat162 c)
Performs __nv_bfloat162 vector fused multiply-add in round-to-nearest-even mode, with saturation to [0.0, 1.0].

**Parameters**
- **a** - __nv_bfloat162. Is only being read.
b
- nv_bfloat16. Is only being read.

c
- nv_bfloat16. Is only being read.

Returns
nv_bfloat16

- The result of elementwise fused multiply-add operation on vectors a, b, and c, with respect to saturation.

Description
Performs nv_bfloat16 vector multiply on inputs a and b, then performs a nv_bfloat16 vector add of the result with c, rounding the result once in round-to-nearest-even mode, and clamps the results to range [0.0, 1.0]. NaN results are flushed to +0.0.

__host____device__hmul2 (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs nv_bfloat16 vector multiplication in round-to-nearest-even mode.

Description
Performs nv_bfloat16 vector multiplication of inputs a and b, in round-to-nearest-even mode.

__host____device__hmul2_rn (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs nv_bfloat16 vector multiplication in round-to-nearest-even mode.

Description
Performs nv_bfloat16 vector multiplication of inputs a and b, in round-to-nearest-even mode. Prevents floating-point contractions of mul+add or sub into fma.

__host____device__hmul2_sat (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs nv_bfloat16 vector multiplication in round-to-nearest-even mode, with saturation to [0.0, 1.0].

Parameters

a
- nv_bfloat16. Is only being read.
nv_bfloat162. Is only being read.

Returns
nv_bfloat162

- The result of elementwise multiplication of vectors a and b, with respect to saturation.

Description
Performs nv_bfloat162 vector multiplication of inputs a and b, in round-to-nearest-even mode, and clamps the results to range [0.0, 1.0]. NaN results are flushed to +0.0.

__host__ __device__ hneg2 (const __nv_bfloat162 a)
Negates both halves of the input nv_bfloat162 number and returns the result.

Description
Negates both halves of the input nv_bfloat162 number a and returns the result.

__host__ __device__ hsub2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs nv_bfloat162 vector subtraction in round-to-nearest-even mode.

Description
Subtracts nv_bfloat162 input vector b from input vector a in round-to-nearest-even mode.

__host__ __device__ hsub2_rn (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs nv_bfloat162 vector subtraction in round-to-nearest-even mode.

Description
Subtracts nv_bfloat162 input vector b from input vector a in round-to-nearest-even mode. Prevents floating-point contractions of mul+sub into fma.
__host__ __device__ hsub2_sat (const __nv_bfloat162 a, const __nv_bfloat162 b)

Performs \texttt{nv\_bfloat162} vector subtraction in round-to-nearest-even mode, with saturation to [0.0, 1.0].

**Parameters**

\texttt{a}
- \texttt{nv\_bfloat162}. Is only being read.

\texttt{b}
- \texttt{nv\_bfloat162}. Is only being read.

**Returns**

\texttt{nv\_bfloat162}
- The subtraction of vector \texttt{b} from \texttt{a}, with respect to saturation.

**Description**

Subtracts \texttt{nv\_bfloat162} input vector \texttt{b} from input vector \texttt{a} in round-to-nearest-even mode, and clamps the results to range [0.0, 1.0]. NaN results are flushed to +0.0.

__device__ atomicAdd (const __nv_bfloat162 *address, const __nv_bfloat162 val)

Vector add \texttt{val} to the value stored at \texttt{address} in global or shared memory, and writes this value back to \texttt{address}. The atomicity of the add operation is guaranteed separately for each of the two \texttt{nv\_bfloat16} elements; the entire \texttt{__nv\_bfloat162} is not guaranteed to be atomic as a single 32-bit access.

**Parameters**

\texttt{address}
- \texttt{__nv\_bfloat162*}. An address in global or shared memory.

\texttt{val}
- \texttt{__nv\_bfloat162}. The value to be added.

**Returns**

\texttt{__nv\_bfloat162}
- The old value read from \texttt{address}. 
Description
The location of address must be in global or shared memory. This operation has undefined behavior otherwise. This operation is natively supported by devices of compute capability 9.x and higher, older devices use emulation path.

Note:
For more details for this function see the Atomic Functions section in the CUDA C++ Programming Guide.

__host__ __device__ operator* (const __nv_bfloat162 lh, const __nv_bfloat162 rh)

Description
Performs packed nv_bfloat16 multiplication operation. See also __hmul2(__nv_bfloat16, __nv_bfloat16)

__host__ __device__ operator*=(__nv_bfloat162 lh, const __nv_bfloat162 rh)

Description
Performs packed nv_bfloat16 compound assignment with multiplication operation.

__host__ __device__ operator+ (const __nv_bfloat162 h)

Description
Implements packed nv_bfloat16 unary plus operator, returns input value.

__host__ __device__ operator+ (const __nv_bfloat162 lh, const __nv_bfloat162 rh)

Description
Performs packed nv_bfloat16 addition operation. See also __hadd2(__nv_bfloat16, __nv_bfloat16)
__host__ __device__ operator++ (__nv_bfloat16 & h, const int ignored)

**Description**

Performs packed nv_bfloat16 postfix increment operation.

__host__ __device__ operator++ (__nv_bfloat16 & h)

**Description**

Performs packed nv_bfloat16 prefix increment operation.

__host__ __device__ operator+= (__nv_bfloat16 & lh, const __nv_bfloat16 & rh)

**Description**

Performs packed nv_bfloat16 compound assignment with addition operation.

__host__ __device__ operator-= (const __nv_bfloat16 & h)

**Description**

Implements packed nv_bfloat16 unary minus operator. See also __hneg2(__nv_bfloat16)

__host__ __device__ operator-= (const __nv_bfloat16 & lh, const __nv_bfloat16 & rh)

**Description**

Performs packed nv_bfloat16 subtraction operation. See also __hsub2(__nv_bfloat16, __nv_bfloat16)

__host__ __device__ operator-- (const __nv_bfloat16 & h, const int ignored)

**Description**

Performs packed nv_bfloat16 postfix decrement operation.
__host____device__ operator-- (__nv_bfloat16 h)

Description
Performs packed __nv_bfloat16 prefix decrement operation.

__host____device__ operator-= (__nv_bfloat16 lh, const __nv_bfloat16 rh)

Description
Performs packed __nv_bfloat16 compound assignment with subtraction operation.

__host____device__ operator/ (const __nv_bfloat16 lh, const __nv_bfloat16 rh)

Description
Performs packed __nv_bfloat16 division operation. See also __h2div (__nv_bfloat16, __nv_bfloat16)

__host____device__ operator/= (__nv_bfloat16 lh, const __nv_bfloat16 rh)

Description
Performs packed __nv_bfloat16 compound assignment with division operation.

1.3.4. Bfloat16 Comparison Functions

Bfloat16 Precision Intrinsics
To use these functions, include the header file cuda_bf16.h in your program.

__host____device__ bool __heq (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs __nv_bfloat16 if-equal comparison.

Parameters

a
- __nv_bfloat16. Is only being read.

b
- __nv_bfloat16. Is only being read.
Returns

bool

- The boolean result of if-equal comparison of a and b.

Description

Performs `nv_bfloat16` if-equal comparison of inputs a and b. NaN inputs generate false results.

```c
__host__ __device__ bool __hequ (const __nv_bfloat16 a,
   const __nv_bfloat16 b)
```

Performs `nv_bfloat16` unordered if-equal comparison.

Parameters

**a**
- `nv_bfloat16`. Is only being read.

**b**
- `nv_bfloat16`. Is only being read.

Returns

bool

- The boolean result of unordered if-equal comparison of a and b.

Description

Performs `nv_bfloat16` if-equal comparison of inputs a and b. NaN inputs generate true results.

```c
__host__ __device__ bool __hge (const __nv_bfloat16 a,
   const __nv_bfloat16 b)
```

Performs `nv_bfloat16` greater-equal comparison.

Parameters

**a**
- `nv_bfloat16`. Is only being read.

**b**
- `nv_bfloat16`. Is only being read.

Returns

bool
The boolean result of greater-equal comparison of \(a\) and \(b\).

**Description**

Performs `nv_bfloat16` greater-equal comparison of inputs \(a\) and \(b\). NaN inputs generate false results.

```c
__host__ __device__ bool __hgeu (const __nv_bfloat16 a,
const __nv_bfloat16 b)
```

Performs `nv_bfloat16` unordered greater-equal comparison.

**Parameters**

\(a\)
- `nv_bfloat16`. Is only being read.

\(b\)
- `nv_bfloat16`. Is only being read.

**Returns**

bool

The boolean result of unordered greater-equal comparison of \(a\) and \(b\).

**Description**

Performs `nv_bfloat16` greater-equal comparison of inputs \(a\) and \(b\). NaN inputs generate true results.

```c
__host__ __device__ bool __hgt (const __nv_bfloat16 a,
const __nv_bfloat16 b)
```

Performs `nv_bfloat16` greater-than comparison.

**Parameters**

\(a\)
- `nv_bfloat16`. Is only being read.

\(b\)
- `nv_bfloat16`. Is only being read.

**Returns**

bool

The boolean result of greater-than comparison of \(a\) and \(b\).
**Description**
Performs `nv_bfloat16` greater-than comparison of inputs `a` and `b`. NaN inputs generate false results.

```c
__host__ __device__ bool __hgtu (const __nv_bfloat16 a, const __nv_bfloat16 b)
```
Performs `nv_bfloat16` unordered greater-than comparison.

**Parameters**
- `a` - `nv_bfloat16`. Is only being read.
- `b` - `nv_bfloat16`. Is only being read.

**Returns**
bool
- The boolean result of unordered greater-than comparison of `a` and `b`.

**Description**
Performs `nv_bfloat16` greater-than comparison of inputs `a` and `b`. NaN inputs generate true results.

```c
__host__ __device__ int __hisinf (const __nv_bfloat16 a)
```
Checks if the input `nv_bfloat16` number is infinite.

**Parameters**
- `a` - `nv_bfloat16`. Is only being read.

**Returns**
int
-1 iff `a` is equal to negative infinity,
1 iff `a` is equal to positive infinity,
0 otherwise.

**Description**
Checks if the input `nv_bfloat16` number `a` is infinite.
__host__ __device__ bool __hisnan (const __nv_bfloat16 a)
Determine whether __nv_bfloat16 argument is a NaN.

**Parameters**

`a`
- __nv_bfloat16. Is only being read.

**Returns**

bool
- true iff argument is NaN.

**Description**

Determine whether __nv_bfloat16 value `a` is a NaN.

__host__ __device__ bool __hle (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs __nv_bfloat16 less-equal comparison.

**Parameters**

`a`
- __nv_bfloat16. Is only being read.

`b`
- __nv_bfloat16. Is only being read.

**Returns**

bool
- The boolean result of less-equal comparison of `a` and `b`.

**Description**

Performs __nv_bfloat16 less-equal comparison of inputs `a` and `b`. NaN inputs generate false results.
__host__ __device__ bool __hleu (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs nv_bfloat16 unordered less-equal comparison.

Parameters

a
- nv_bfloat16. Is only being read.

b
- nv_bfloat16. Is only being read.

Returns

bool
- The boolean result of unordered less-equal comparison of a and b.

Description

Performs nv_bfloat16 less-equal comparison of inputs a and b. NaN inputs generate true results.

__host__ __device__ bool __hlt (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs nv_bfloat16 less-than comparison.

Parameters

a
- nv_bfloat16. Is only being read.

b
- nv_bfloat16. Is only being read.

Returns

bool
- The boolean result of less-than comparison of a and b.

Description

Performs nv_bfloat16 less-than comparison of inputs a and b. NaN inputs generate false results.
__host__ __device__ bool __hltu (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs \(\text{nv\_bfloat16}\) unordered less-than comparison.

Parameters

a
- \(\text{nv\_bfloat16}\). Is only being read.

b
- \(\text{nv\_bfloat16}\). Is only being read.

Returns

bool

- The boolean result of unordered less-than comparison of \(a\) and \(b\).

Description

Performs \(\text{nv\_bfloat16}\) less-than comparison of inputs \(a\) and \(b\). NaN inputs generate true results.

__host__ __device__ __hmax (const __nv_bfloat16 a, const __nv_bfloat16 b)
Calculates \(\text{nv\_bfloat16}\) maximum of two input values.

Description

Calculates \(\text{nv\_bfloat16}\) \(\max(a, b)\) defined as \(a > b\) ? \(a\) : \(b\).

- If either of inputs is NaN, the other input is returned.
- If both inputs are NaNs, then canonical NaN is returned.
- If values of both inputs are 0.0, then +0.0 > -0.0

__host__ __device__ __hmax_nan (const __nv_bfloat16 a, const __nv_bfloat16 b)
Calculates \(\text{nv\_bfloat16}\) maximum of two input values, NaNs pass through.

Description

Calculates \(\text{nv\_bfloat16}\) \(\max(a, b)\) defined as \(a > b\) ? \(a\) : \(b\).

- If either of inputs is NaN, then canonical NaN is returned.
- If values of both inputs are 0.0, then +0.0 > -0.0
__host__ __device__ hmin (const __nv_bfloat16 a, const __nv_bfloat16 b)
Calculates nv_bfloat16 minimum of two input values.

Description
Calculates \( \text{nv\_bfloat16 min}(a, b) \) defined as \( (a < b) \ ? a : b \).

- If either of inputs is NaN, the other input is returned.
- If both inputs are NaNs, then canonical NaN is returned.
- If values of both inputs are 0.0, then +0.0 > -0.0

__host__ __device__ hmin_nan (const __nv_bfloat16 a, const __nv_bfloat16 b)
Calculates nv_bfloat16 minimum of two input values, NaNs pass through.

Description
Calculates \( \text{nv\_bfloat16 min}(a, b) \) defined as \( (a < b) \ ? a : b \).

- If either of inputs is NaN, then canonical NaN is returned.
- If values of both inputs are 0.0, then +0.0 > -0.0

__host__ __device__ bool __hne (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs nv_bfloat16 not-equal comparison.

Parameters
- \( a \) - nv_bfloat16. Is only being read.
- \( b \) - nv_bfloat16. Is only being read.

Returns
- bool

- The boolean result of not-equal comparison of \( a \) and \( b \).

Description
Performs \( \text{nv\_bfloat16 not-equal comparison} \) of inputs \( a \) and \( b \). NaN inputs generate false results.
__host__ __device__ bool __hneu (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs nv_bfloat16 unordered not-equal comparison.

Parameters

a
- nv_bfloat16. Is only being read.

b
- nv_bfloat16. Is only being read.

Returns

bool
- The boolean result of unordered not-equal comparison of a and b.

Description

Performs nv_bfloat16 not-equal comparison of inputs a and b. NaN inputs generate true results.

__host__ __device__ __CUDA_BF16_FORCEINLINE__ bool operator!= (const __nv_bfloat16 lh, const __nv_bfloat16 rh)

Description

Performs nv_bfloat16 unordered compare not-equal operation. See also __hneu(__nv_bfloat16, __nv_bfloat16)

__host__ __device__ __CUDA_BF16_FORCEINLINE__ bool operator< (const __nv_bfloat16 lh, const __nv_bfloat16 rh)

Description

Performs nv_bfloat16 ordered less-than compare operation. See also __hlt(__nv_bfloat16, __nv_bfloat16)
__host__ __device__ __CUDA_BF16_FORCEINLINE__ bool operator<=(const __nv_bfloat16 lh, const __nv_bfloat16 rh)

Description
Perform \texttt{nv.bfloat16} ordered less-or-equal compare operation. See also \_hle\(_{\texttt{nv.bfloat16, nv.bfloat16}}\)

__host__ __device__ __CUDA_BF16_FORCEINLINE__ bool operator==(const __nv_bfloat16 lh, const __nv_bfloat16 rh)

Description
Perform \texttt{nv.bfloat16} ordered compare equal operation. See also \_heq\(_{\texttt{nv.bfloat16, nv.bfloat16}}\)

__host__ __device__ __CUDA_BF16_FORCEINLINE__ bool operator>(const __nv_bfloat16 lh, const __nv_bfloat16 rh)

Description
Perform \texttt{nv.bfloat16} ordered greater-than compare operation. See also \_hgt\(_{\texttt{nv.bfloat16, nv.bfloat16}}\)

__host__ __device__ __CUDA_BF16_FORCEINLINE__ bool operator>=(const __nv_bfloat16 lh, const __nv_bfloat16 rh)

Description
Perform \texttt{nv.bfloat16} ordered greater-or-equal compare operation. See also \_hge\(_{\texttt{nv.bfloat16, nv.bfloat16}}\)

1.3.5. Bfloat16 Comparison Functions

Bfloat16 Precision Intrinsics

To use these functions, include the header file \texttt{cuda_bf16.h} in your program.
__host____device__ bool __hbeq2 (const __nv_bfloat162 a, const __nv_bfloat162 b)

Performs __nv_bfloat162 vector if-equal comparison and returns boolean true iff both __nv_bfloat16 results are true, boolean false otherwise.

Parameters

a - __nv_bfloat162. Is only being read.

b - __nv_bfloat162. Is only being read.

Returns

bool

- true if both __nv_bfloat16 results of if-equal comparison of vectors a and b are true;
- false otherwise.

Description

Performs __nv_bfloat162 vector if-equal comparison of inputs a and b. The bool result is set to true only if both __nv_bfloat16 if-equal comparisons evaluate to true, or false otherwise. NaN inputs generate false results.

__host____device__ bool __hbequ2 (const __nv_bfloat162 a, const __nv_bfloat162 b)

Performs __nv_bfloat162 vector unordered if-equal comparison and returns boolean true iff both __nv_bfloat16 results are true, boolean false otherwise.

Parameters

a - __nv_bfloat162. Is only being read.

b - __nv_bfloat162. Is only being read.

Returns

bool

- true if both __nv_bfloat16 results of unordered if-equal comparison of vectors a and b are true;
- false otherwise.
Description
Performs \texttt{nv\_bfloat162} vector if-equal comparison of inputs \texttt{a} and \texttt{b}. The bool result is set to true only if both \texttt{nv\_bfloat16} if-equal comparisons evaluate to true, or false otherwise. NaN inputs generate true results.

\texttt{__host\_\_device\_\_bool \_\_hbge2 (const \_\_nv\_\_bfloat162 \_\_a, const \_\_nv\_\_bfloat162 \_\_b)}

Performs \texttt{nv\_bfloat162} vector greater-equal comparison and returns boolean true iff both \texttt{nv\_bfloat16} results are true, boolean false otherwise.

Parameters
\begin{itemize}
  \item \texttt{a} \\
    - \texttt{nv\_bfloat162}. Is only being read.
  \item \texttt{b} \\
    - \texttt{nv\_bfloat162}. Is only being read.
\end{itemize}

Returns
\begin{itemize}
  \item \texttt{bool} \\
    \begin{itemize}
      \item true if both \texttt{nv\_bfloat16} results of greater-equal comparison of vectors \texttt{a} and \texttt{b} are true;
      \item false otherwise.
    \end{itemize}
\end{itemize}

Description
Performs \texttt{nv\_bfloat162} vector greater-equal comparison of inputs \texttt{a} and \texttt{b}. The bool result is set to true only if both \texttt{nv\_bfloat16} greater-equal comparisons evaluate to true, or false otherwise. NaN inputs generate false results.

\texttt{__host\_\_device\_\_bool \_\_hbgeu2 (const \_\_nv\_\_bfloat162 \_\_a, const \_\_nv\_\_bfloat162 \_\_b)}

Performs \texttt{nv\_bfloat162} vector unordered greater-equal comparison and returns boolean true iff both \texttt{nv\_bfloat16} results are true, boolean false otherwise.

Parameters
\begin{itemize}
  \item \texttt{a} \\
    - \texttt{nv\_bfloat162}. Is only being read.
  \item \texttt{b} \\
    - \texttt{nv\_bfloat162}. Is only being read.
\end{itemize}
Returns

bool

- true if both \texttt{nv\_bfloa16} results of unordered greater-equal comparison of vectors \texttt{a} and \texttt{b} are true;
- false otherwise.

Description

Performs \texttt{nv\_bfloa16} vector greater-equal comparison of inputs \texttt{a} and \texttt{b}. The bool result is set to true only if both \texttt{nv\_bfloa16} greater-equal comparisons evaluate to true, or false otherwise. NaN inputs generate true results.

\texttt{\_\_host\_\_device\_\_bool \_\_hbgt2 (const \_\_nv\_bfloa16 a, const \_\_nv\_bfloa16 b)}

Performs \texttt{nv\_bfloa16} vector greater-than comparison and returns boolean true iff both \texttt{nv\_bfloa16} results are true, boolean false otherwise.

Parameters

\texttt{a}
- \texttt{nv\_bfloa16}. Is only being read.

\texttt{b}
- \texttt{nv\_bfloa16}. Is only being read.

Returns

bool

- true if both \texttt{nv\_bfloa16} results of greater-than comparison of vectors \texttt{a} and \texttt{b} are true;
- false otherwise.

Description

Performs \texttt{nv\_bfloa16} vector greater-than comparison of inputs \texttt{a} and \texttt{b}. The bool result is set to true only if both \texttt{nv\_bfloa16} greater-than comparisons evaluate to true, or false otherwise. NaN inputs generate false results.
__host____device__ bool __hbgtu2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs nv_bfloat162 vector unordered greater-than comparison and returns boolean true iff both nv_bfloat16 results are true, boolean false otherwise.

Parameters
a
- nv_bfloat162. Is only being read.
b
- nv_bfloat162. Is only being read.

Returns
bool

- true if both nv_bfloat16 results of unordered greater-than comparison of vectors a and b are true;
- false otherwise.

Description
Performs nv_bfloat162 vector greater-than comparison of inputs a and b. The bool result is set to true only if both nv_bfloat16 greater-than comparisons evaluate to true, or false otherwise. NaN inputs generate true results.

__host____device__ bool __hble2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs nv_bfloat162 vector less-equal comparison and returns boolean true iff both nv_bfloat16 results are true, boolean false otherwise.

Parameters
a
- nv_bfloat162. Is only being read.
b
- nv_bfloat162. Is only being read.

Returns
bool

- true if both nv_bfloat16 results of less-equal comparison of vectors a and b are true;
- false otherwise.
Description

Performs \texttt{nv\_bfloat162} vector less-equal comparison of inputs \(a\) and \(b\). The bool result is set to true only if both \texttt{nv\_bfloat16} less-equal comparisons evaluate to true, or false otherwise. NaN inputs generate false results.

\begin{verbatim}
__host____device__ bool __hbleu2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
\end{verbatim}

Performs \texttt{nv\_bfloat162} vector unordered less-equal comparison and returns boolean true iff both \texttt{nv\_bfloat16} results are true, boolean false otherwise.

Parameters

\begin{itemize}
  \item \texttt{a} - \texttt{nv\_bfloat162}. Is only being read.
  \item \texttt{b} - \texttt{nv\_bfloat162}. Is only being read.
\end{itemize}

Returns

\begin{verbatim}
bool
\end{verbatim}

\begin{itemize}
  \item true if both \texttt{nv\_bfloat16} results of unordered less-equal comparison of vectors \(a\) and \(b\) are true;
  \item false otherwise.
\end{itemize}

Description

Performs \texttt{nv\_bfloat162} vector less-equal comparison of inputs \(a\) and \(b\). The bool result is set to true only if both \texttt{nv\_bfloat16} less-equal comparisons evaluate to true, or false otherwise. NaN inputs generate true results.

\begin{verbatim}
__host____device__ bool __hblt2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
\end{verbatim}

Performs \texttt{nv\_bfloat162} vector less-than comparison and returns boolean true iff both \texttt{nv\_bfloat16} results are true, boolean false otherwise.

Parameters

\begin{itemize}
  \item \texttt{a} - \texttt{nv\_bfloat162}. Is only being read.
  \item \texttt{b} - \texttt{nv\_bfloat162}. Is only being read.
\end{itemize}
Returns

bool

- true if both `nv_bfloat16` results of less-than comparison of vectors `a` and `b` are true;
- false otherwise.

Description

Performs `nv_bfloat16` vector less-than comparison of inputs `a` and `b`. The bool result is set to true only if both `nv_bfloat16` less-than comparisons evaluate to true, or false otherwise. NaN inputs generate false results.

```
__host__ __device__ bool __hbltu2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
```

Performs `nv_bfloat16` vector unordered less-than comparison and returns boolean true iff both `nv_bfloat16` results are true, boolean false otherwise.

Parameters

- `a` - `nv_bfloat162`. Is only being read.
- `b` - `nv_bfloat162`. Is only being read.

Returns

bool

- true if both `nv_bfloat16` results of unordered less-than comparison of vectors `a` and `b` are true;
- false otherwise.

Description

Performs `nv_bfloat16` vector less-than comparison of inputs `a` and `b`. The bool result is set to true only if both `nv_bfloat16` less-than comparisons evaluate to true, or false otherwise. NaN inputs generate true results.
__host____device__ bool __hbne2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs __nv_bfloat162 vector not-equal comparison and returns boolean true iff both __nv_bfloat16 results are true, boolean false otherwise.

Parameters
a
- __nv_bfloat162. Is only being read.

b
- __nv_bfloat162. Is only being read.

Returns
bool
- true if both __nv_bfloat16 results of not-equal comparison of vectors a and b are true,
- false otherwise.

Description
Performs __nv_bfloat162 vector not-equal comparison of inputs a and b. The bool result is set to true only if both __nv_bfloat16 not-equal comparisons evaluate to true, or false otherwise. NaN inputs generate false results.

__host____device__ bool __hbneu2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs __nv_bfloat162 vector unordered not-equal comparison and returns boolean true iff both __nv_bfloat16 results are true, boolean false otherwise.

Parameters
a
- __nv_bfloat162. Is only being read.

b
- __nv_bfloat162. Is only being read.

Returns
bool
- true if both __nv_bfloat16 results of unordered not-equal comparison of vectors a and b are true;
- false otherwise.
Description
Performs \texttt{nv\_bfloat162} vector not-equal comparison of inputs \texttt{a} and \texttt{b}. The bool result is set to true only if both \texttt{nv\_bfloat16} not-equal comparisons evaluate to true, or false otherwise. NaN inputs generate true results.

\texttt{__host\_\_device\_\_heq2 (const \_\_nv\_bfloat162 a, const \_\_nv\_bfloat162 b)}
Performs \texttt{nv\_bfloat162} vector if-equal comparison.

Parameters
\texttt{a}
- \texttt{nv\_bfloat162}. Is only being read.
\texttt{b}
- \texttt{nv\_bfloat162}. Is only being read.

Returns
\texttt{nv\_bfloat162}
\begin{itemize}
\item The vector result of if-equal comparison of vectors \texttt{a} and \texttt{b}.
\end{itemize}

Description
Performs \texttt{nv\_bfloat162} vector if-equal comparison of inputs \texttt{a} and \texttt{b}. The corresponding \texttt{nv\_bfloat16} results are set to 1.0 for true, or 0.0 for false. NaN inputs generate false results.

\texttt{__host\_\_device\_\_unsigned\_\_heq2\_\_mask (const \_\_nv\_bfloat162 a, const \_\_nv\_bfloat162 b)}
Performs \texttt{nv\_bfloat162} vector if-equal comparison.

Parameters
\texttt{a}
- \texttt{nv\_bfloat162}. Is only being read.
\texttt{b}
- \texttt{nv\_bfloat162}. Is only being read.

Returns
\texttt{unsigned int}
\begin{itemize}
\item The vector mask result of if-equal comparison of vectors \texttt{a} and \texttt{b}.
\end{itemize}
Description
Performs `nv_bfloat162` vector if-equal comparison of inputs `a` and `b`. The corresponding unsigned bits are set to 0xFFFF for true, or 0x0 for false. NaN inputs generate false results.

```c
__host__ __device__ __hequ2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
```
Performs `nv_bfloat162` vector unordered if-equal comparison.

Parameters
a
- `nv_bfloat162`. Is only being read.
b
- `nv_bfloat162`. Is only being read.

Returns
`nv_bfloat162`
- The vector result of unordered if-equal comparison of vectors `a` and `b`.

Description
Performs `nv_bfloat162` vector if-equal comparison of inputs `a` and `b`. The corresponding `nv_bfloat16` results are set to 1.0 for true, or 0.0 for false. NaN inputs generate true results.

```c
__host__ __device__ unsigned __hequ2_mask (const __nv_bfloat162 a, const __nv_bfloat162 b)
```
Performs `nv_bfloat162` vector unordered if-equal comparison.

Parameters
a
- `nv_bfloat162`. Is only being read.
b
- `nv_bfloat162`. Is only being read.

Returns
unsigned int
- The vector mask result of unordered if-equal comparison of vectors `a` and `b`.
Description
Performs `nv_bfloat162` vector if-equal comparison of inputs `a` and `b`. The corresponding unsigned bits are set to 0xFFFF for true, or 0x0 for false. NaN inputs generate true results.

```c
__host__ __device__ hge2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
```
Performs `nv_bfloat162` vector greater-equal comparison.

Parameters
- `a`: `nv_bfloat162`. Is only being read.
- `b`: `nv_bfloat162`. Is only being read.

Returns
- `nv_bfloat162`:
  - The vector result of greater-equal comparison of vectors `a` and `b`.

Description
Performs `nv_bfloat162` vector greater-equal comparison of inputs `a` and `b`. The corresponding `nv_bfloat16` results are set to 1.0 for true, or 0.0 for false. NaN inputs generate false results.

```c
__host__ __device__ unsigned __hge2_mask (const __nv_bfloat162 a, const __nv_bfloat162 b)
```
Performs `nv_bfloat162` vector greater-equal comparison.

Parameters
- `a`: `nv_bfloat162`. Is only being read.
- `b`: `nv_bfloat162`. Is only being read.

Returns
- `unsigned int`:
  - The vector mask result of greater-equal comparison of vectors `a` and `b`. 
Description
Performs `nv_bfloat16` vector greater-equal comparison of inputs `a` and `b`. The corresponding `unsigned` bits are set to `0xFFFF` for true, or `0x0` for false. NaN inputs generate false results.

__host__ __device__ `hgeu2` (const `__nv_bfloat16` `a`, const `__nv_bfloat16` `b`)
Performs `nv_bfloat16` vector unordered greater-equal comparison.

Parameters
- `a` - `nv_bfloat16`. Is only being read.
- `b` - `nv_bfloat16`. Is only being read.

Returns
`nv_bfloat16`
- The `nv_bfloat16` vector result of unordered greater-equal comparison of vectors `a` and `b`.

Description
Performs `nv_bfloat16` vector greater-equal comparison of inputs `a` and `b`. The corresponding `nv_bfloat16` results are set to `1.0` for true, or `0.0` for false. NaN inputs generate true results.

__host__ __device__ `unsigned __hgeu2_mask` (const `__nv_bfloat16` `a`, const `__nv_bfloat16` `b`)
Performs `nv_bfloat16` vector unordered greater-equal comparison.

Parameters
- `a` - `nv_bfloat16`. Is only being read.
- `b` - `nv_bfloat16`. Is only being read.

Returns
`unsigned int`
- The vector mask result of unordered greater-equal comparison of vectors `a` and `b`. 
Description

Performs `nv_bfloat162` vector greater-equal comparison of inputs `a` and `b`. The corresponding `unsigned` bits are set to 0xFFFF for true, or 0x0 for false. NaN inputs generate true results.

```c
__host__ __device__ __hgt2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
```

Performs `nv_bfloat162` vector greater-than comparison.

Parameters

- **a**
  - `nv_bfloat162`. Is only being read.
- **b**
  - `nv_bfloat162`. Is only being read.

Returns

`nv_bfloat162`

- The vector result of greater-than comparison of vectors `a` and `b`.

Description

Performs `nv_bfloat162` vector greater-than comparison of inputs `a` and `b`. The corresponding `nv_bfloat16` results are set to 1.0 for true, or 0.0 for false. NaN inputs generate false results.

```c
__host__ __device__ unsigned __hgt2_mask (const __nv_bfloat162 a, const __nv_bfloat162 b)
```

Performs `nv_bfloat162` vector greater-than comparison.

Parameters

- **a**
  - `nv_bfloat162`. Is only being read.
- **b**
  - `nv_bfloat162`. Is only being read.

Returns

`unsigned int`

- The vector mask result of greater-than comparison of vectors `a` and `b`. 
Description
Performs `nv_bfloat162` vector greater-than comparison of inputs `a` and `b`. The corresponding `unsigned` bits are set to 0xFFFF for true, or 0x0 for false. NaN inputs generate false results.

```
__host__ __device__ hgtu2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
```
Performs `nv_bfloat162` vector unordered greater-than comparison.

Parameters

- **a** - `nv_bfloat162`. Is only being read.
- **b** - `nv_bfloat162`. Is only being read.

Returns

`nv_bfloat162`

- The `nv_bfloat162` vector result of unordered greater-than comparison of vectors `a` and `b`.

Description
Performs `nv_bfloat162` vector greater-than comparison of inputs `a` and `b`. The corresponding `nv_bfloat16` results are set to 1.0 for true, or 0.0 for false. NaN inputs generate true results.

```
__host__ __device__ unsigned __hgtu2_mask (const __nv_bfloat162 a, const __nv_bfloat162 b)
```
Performs `nv_bfloat162` vector unordered greater-than comparison.

Parameters

- **a** - `nv_bfloat162`. Is only being read.
- **b** - `nv_bfloat162`. Is only being read.

Returns

`unsigned int`

- The vector mask result of unordered greater-than comparison of vectors `a` and `b`. 
Description
Performs \texttt{nv\_bfloat162} vector greater-than comparison of inputs \texttt{a} and \texttt{b}. The corresponding unsigned bits are set to 0xFFFF for true, or 0x0 for false. NaN inputs generate true results.

\[\texttt{__host\_device\_hisnan2 (const \_nv\_bfloat162 a)}\]
Determine whether \texttt{nv\_bfloat162} argument is a NaN.

Parameters
\texttt{a}
- \texttt{nv\_bfloat162}. Is only being read.

Returns
\texttt{nv\_bfloat162}
\- The \texttt{nv\_bfloat162} with the corresponding \texttt{nv\_bfloat16} results set to 1.0 for NaN, 0.0 otherwise.

Description
Determine whether each \texttt{nv\_bfloat} of input \texttt{nv\_bfloat162} number \texttt{a} is a NaN.

\[\texttt{__host\_device\_hle2 (const \_nv\_bfloat162 a, const \_nv\_bfloat162 b)}\]
Performs \texttt{nv\_bfloat162} vector less-equal comparison.

Parameters
\texttt{a}
- \texttt{nv\_bfloat162}. Is only being read.
\texttt{b}
- \texttt{nv\_bfloat162}. Is only being read.

Returns
\texttt{nv\_bfloat162}
\- The \texttt{nv\_bfloat162} result of less-equal comparison of vectors \texttt{a} and \texttt{b}.

Description
Performs \texttt{nv\_bfloat162} vector less-equal comparison of inputs \texttt{a} and \texttt{b}. The corresponding \texttt{nv\_bfloat16} results are set to 1.0 for true, or 0.0 for false. NaN inputs generate false results.
__host__ __device__ unsigned __hle2_mask (const __nv_bfloat162 a, const __nv_bfloat162 b)

Performs nv_bfloat162 vector less-equal comparison.

Parameters

a
- nv_bfloat162. Is only being read.

b
- nv_bfloat162. Is only being read.

Returns

unsigned int
- The vector mask result of less-equal comparison of vectors a and b.

Description

Performs nv_bfloat162 vector less-equal comparison of inputs a and b. The corresponding unsigned bits are set to 0xFFFF for true, or 0x0 for false. NaN inputs generate false results.

__host__ __device__ __hleu2 (const __nv_bfloat162 a, const __nv_bfloat162 b)

Performs nv_bfloat16 vector unordered less-equal comparison.

Parameters

a
- nv_bfloat162. Is only being read.

b
- nv_bfloat162. Is only being read.

Returns

nv_bfloat162
- The vector result of unordered less-equal comparison of vectors a and b.

Description

Performs nv_bfloat16 vector less-equal comparison of inputs a and b. The corresponding nv_bfloat16 results are set to 1.0 for true, or 0.0 for false. NaN inputs generate true results.
__host____device__ unsigned __hleu2_mask (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs `nv_bfloat162` vector unordered less-equal comparison.

Parameters

a
- `nv_bfloat162`. Is only being read.

b
- `nv_bfloat162`. Is only being read.

Returns

unsigned int
- The vector mask result of unordered less-equal comparison of vectors a and b.

Description

Performs `nv_bfloat162` vector less-equal comparison of inputs a and b. The corresponding unsigned bits are set to 0xFFFF for true, or 0x0 for false. NaN inputs generate true results.

__host____device____hlt2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs `nv_bfloat162` vector less-than comparison.

Parameters

a
- `nv_bfloat162`. Is only being read.

b
- `nv_bfloat162`. Is only being read.

Returns

nv_bfloat162
- The `nv_bfloat162` vector result of less-than comparison of vectors a and b.

Description

Performs `nv_bfloat162` vector less-than comparison of inputs a and b. The corresponding `nv_bfloat16` results are set to 1.0 for true, or 0.0 for false. NaN inputs generate false results.
__host__ __device__ unsigned __hlt2_mask (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs nv_bfloat162 vector less-than comparison.

Parameters
a
- nv_bfloat162. Is only being read.
b
- nv_bfloat162. Is only being read.

Returns
unsigned int

- The vector mask result of less-than comparison of vectors a and b.

Description
Performs nv_bfloat162 vector less-than comparison of inputs a and b. The corresponding unsigned bits are set to 0xFFFF for true, or 0x0 for false. NaN inputs generate false results.

__host__ __device__ hltu2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs nv_bfloat162 vector unordered less-than comparison.

Parameters
a
- nv_bfloat162. Is only being read.
b
- nv_bfloat162. Is only being read.

Returns
nv_bfloat162

- The vector result of unordered less-than comparison of vectors a and b.

Description
Performs nv_bfloat162 vector less-than comparison of inputs a and b. The corresponding nv_bfloat16 results are set to 1.0 for true, or 0.0 for false. NaN inputs generate true results.
__host___device__ unsigned __hltu2_mask (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs nv_bfloat162 vector unordered less-than comparison.

Parameters

a
- nv_bfloat162. Is only being read.

b
- nv_bfloat162. Is only being read.

Returns

unsigned int
- The vector mask result of unordered less-than comparison of vectors a and b.

Description

Performs nv_bfloat162 vector less-than comparison of inputs a and b. The corresponding unsigned bits are set to 0xFFFF for true, or 0x0 for false. NaN inputs generate true results.

__host___device____hmax2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
Calculates nv_bfloat162 vector maximum of two inputs.

Description

Calculates nv_bfloat162 vector max(a, b). Elementwise nv_bfloat16 operation is defined as (a > b) ? a : b.

- If either of inputs is NaN, the other input is returned.
- If both inputs are NaNs, then canonical NaN is returned.
- If values of both inputs are 0.0, then +0.0 > -0.0
- The result of elementwise maximum of vectors a and b
__host__ __device__ hmax2_nan (const __nv_bfloat162 a, const __nv_bfloat162 b)
Calculates nv_bfloat162 vector maximum of two inputs, NaNs pass through.

Description
Calculates nv_bfloat162 vector max(a, b). Elementwise nv_bfloat16 operation is defined as (a > b) ? a : b.

‣ If either of inputs is NaN, then canonical NaN is returned.
‣ If values of both inputs are 0.0, then +0.0 > -0.0
‣ The result of elementwise maximum of vectors a and b, with NaNs pass through

__host__ __device__ hmin2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
Calculates nv_bfloat162 vector minimum of two inputs.

Description
Calculates nv_bfloat162 vector min(a, b). Elementwise nv_bfloat16 operation is defined as (a < b) ? a : b.

‣ If either of inputs is NaN, the other input is returned.
‣ If both inputs are NaNs, then canonical NaN is returned.
‣ If values of both inputs are 0.0, then +0.0 > -0.0
‣ The result of elementwise minimum of vectors a and b

__host__ __device__ hmin2_nan (const __nv_bfloat162 a, const __nv_bfloat162 b)
Calculates nv_bfloat162 vector minimum of two inputs, NaNs pass through.

Description
Calculates nv_bfloat162 vector min(a, b). Elementwise nv_bfloat16 operation is defined as (a < b) ? a : b.

‣ If either of inputs is NaN, then canonical NaN is returned.
‣ If values of both inputs are 0.0, then +0.0 > -0.0
‣ The result of elementwise minimum of vectors a and b, with NaNs pass through
__host__ __device__ hne2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs __nv_bfloat162 vector not-equal comparison.

Parameters

a
- __nv_bfloat162. Is only being read.

b
- __nv_bfloat162. Is only being read.

Returns

__nv_bfloat162

> The vector result of not-equal comparison of vectors a and b.

Description

Performs __nv_bfloat162 vector not-equal comparison of inputs a and b. The corresponding __nv_bfloat16 results are set to 1.0 for true, or 0.0 for false. NaN inputs generate false results.

__host__ __device__ unsigned __hne2_mask (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs __nv_bfloat162 vector not-equal comparison.

Parameters

a
- __nv_bfloat162. Is only being read.

b
- __nv_bfloat162. Is only being read.

Returns

unsigned int

> The vector mask result of not-equal comparison of vectors a and b.

Description

Performs __nv_bfloat162 vector not-equal comparison of inputs a and b. The corresponding unsigned bits are set to 0xFFFF for true, or 0x0 for false. NaN inputs generate false results.
__host__ __device__ hneu2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs \texttt{nv\_bfloat162} vector unordered not-equal comparison.

**Parameters**

- `a` - \texttt{nv\_bfloat162}. Is only being read.
- `b` - \texttt{nv\_bfloat162}. Is only being read.

**Returns**

- \texttt{nv\_bfloat162}
  - The vector result of unordered not-equal comparison of vectors `a` and `b`.

**Description**

Performs \texttt{nv\_bfloat162} vector not-equal comparison of inputs `a` and `b`. The corresponding \texttt{nv\_bfloat16} results are set to 1.0 for true, or 0.0 for false. NaN inputs generate true results.

__host__ __device__ unsigned __hneu2\_mask (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs \texttt{nv\_bfloat162} vector unordered not-equal comparison.

**Parameters**

- `a` - \texttt{nv\_bfloat162}. Is only being read.
- `b` - \texttt{nv\_bfloat162}. Is only being read.

**Returns**

- \texttt{unsigned int}
  - The vector mask result of unordered not-equal comparison of vectors `a` and `b`.

**Description**

Performs \texttt{nv\_bfloat162} vector not-equal comparison of inputs `a` and `b`. The corresponding unsigned bits are set to 0xFFFF for true, or 0x0 for false. NaN inputs generate true results.
__host____device__ __CUDA_BF16_FORCINLINE__ bool 
operator!=(const __nv_bfloat162 lh, const __nv_bfloat162 rh)

Description
Performs packed nv_bfloat16 unordered compare not-equal operation. See also __hbneu2( __nv_bfloat162, __nv_bfloat162)

__host____device__ __CUDA_BF16_FORCINLINE__ bool 
operator<(const __nv_bfloat162 lh, const __nv_bfloat162 rh)

Description
Performs packed nv_bfloat16 ordered less-than compare operation. See also __hblt2( __nv_bfloat162, __nv_bfloat162)

__host____device__ __CUDA_BF16_FORCINLINE__ bool 
operator<=(const __nv_bfloat162 lh, const __nv_bfloat162 rh)

Description
Performs packed nv_bfloat16 ordered less-or-equal compare operation. See also __hble2( __nv_bfloat162, __nv_bfloat162)

__host____device__ __CUDA_BF16_FORCINLINE__ bool 
operator==(const __nv_bfloat162 lh, const __nv_bfloat162 rh)

Description
Performs packed nv_bfloat16 ordered compare equal operation. See also __hbeq2( __nv_bfloat162, __nv_bfloat162)
__host__ __device__ __CUDA_BF16_FORCEINLINE__ bool operator> (const __nv_bfloat162 lh, const __nv_bfloat162 rh)

Description
Performs packed nv_bfloat16 ordered greater-than compare operation. See also __hbgt2( __nv_bfloat162, __nv_bfloat162)

__host__ __device__ __CUDA_BF16_FORCEINLINE__ bool operator>=( const __nv_bfloat162 lh, const __nv_bfloat162 rh)

Description
Performs packed nv_bfloat16 ordered greater-or-equal compare operation. See also __hbge2( __nv_bfloat162, __nv_bfloat162)

1.3.6. Bfloat16 Precision Conversion and Data Movement

Bfloat16 Precision Intrinsics
To use these functions, include the header file cuda_bf16.h in your program.

__host____device__ float2 __bfloat1622float2 (const __nv_bfloat162 a)
Converts both halves of nv_bfloat162 to float2 and returns the result.

Parameters

a
- nv_bfloat162. Is only being read.

Returns
float2
- a converted to float2.

Description
Converts both halves of nv_bfloat162 input a to float2 and returns the result.
__host__ __device__ __bfloat162bfloat162 (const __nv_bfloat16 a)

Returns `nv_bfloat162` with both halves equal to the input value.

Parameters

**a**

- `nv_bfloat16`. Is only being read.

Returns

`nv_bfloat162`

- The vector which has both its halves equal to the input `a`.

Description

Returns `nv_bfloat162` number with both halves equal to the input `nv_bfloat16` number.

__host__ __device__ signed char __bfloat162char_rz (const __nv_bfloat16 h)

Convert a `nv_bfloat16` to a signed char in round-towards-zero mode.

Parameters

**h**

- `nv_bfloat16`. Is only being read.

Returns

signed char

- `h` converted to a signed char.

Description

Convert the `nv_bfloat16` floating-point value `h` to a signed char in round-towards-zero mode. NaN inputs are converted to 0.
__host__ __device__ float __bfloat162float (const __nv_bfloat16 a)
Converts __nv_bfloat16 number to float.

Parameters

\( a \)
- float. Is only being read.

Returns

float
- \( a \) converted to float.

Description

Converts __nv_bfloat16 number \( a \) to float.

__device__ int __bfloat162int_rd (const __nv_bfloat16 h)
Convert a __nv_bfloat16 to a signed integer in round-down mode.

Parameters

\( h \)
- __nv_bfloat16. Is only being read.

Returns

int
- \( h \) converted to a signed integer.

Description

Convert the __nv_bfloat16 floating-point value \( h \) to a signed integer in round-down mode. NaN inputs are converted to 0.

__device__ int __bfloat162int_rn (const __nv_bfloat16 h)
Convert a __nv_bfloat16 to a signed integer in round-to-nearest-even mode.

Parameters

\( h \)
- __nv_bfloat16. Is only being read.
Returns
int
  h converted to a signed integer.

Description
Convert the `nv_bfloat16` floating-point value `h` to a signed integer in round-to-nearest-even mode. NaN inputs are converted to 0.

__device__ int __bfloat162int_ru (const __nv_bfloat16 h)
Convert a `nv_bfloat16` to a signed integer in round-up mode.

Parameters
h
  - `nv_bfloat16`. Is only being read.

Returns
int
  h converted to a signed integer.

Description
Convert the `nv_bfloat16` floating-point value `h` to a signed integer in round-up mode. NaN inputs are converted to 0.

__host____device__ int __bfloat162int_rz (const __nv_bfloat16 h)
Convert a `nv_bfloat16` to a signed integer in round-towards-zero mode.

Parameters
h
  - `nv_bfloat16`. Is only being read.

Returns
int
  h converted to a signed integer.
Description
Convert the `nv_bfloat16` floating-point value `h` to a signed integer in round-towards-zero mode. NaN inputs are converted to 0.

```
__device__ long long int __bfloat162ll_rd (const __nv_bfloat16 h)
```

Convert a `nv_bfloat16` to a signed 64-bit integer in round-down mode.

Parameters
- `h` - `nv_bfloat16`. Is only being read.

Returns
- long long int
  - `h` converted to a signed 64-bit integer.

Description
Convert the `nv_bfloat16` floating-point value `h` to a signed 64-bit integer in round-down mode. NaN inputs return a long long int with hex value of 0x8000000000000000.

```
__device__ long long int __bfloat162ll_rn (const __nv_bfloat16 h)
```

Convert a `nv_bfloat16` to a signed 64-bit integer in round-to-nearest-even mode.

Parameters
- `h` - `nv_bfloat16`. Is only being read.

Returns
- long long int
  - `h` converted to a signed 64-bit integer.

Description
Convert the `nv_bfloat16` floating-point value `h` to a signed 64-bit integer in round-to-nearest-even mode. NaN inputs return a long long int with hex value of 0x8000000000000000.
__device__ long long int __bfloat162ll_ru (const __nv_bfloat16 h)
Convert a nv_bfloat16 to a signed 64-bit integer in round-up mode.

Parameters

h
- nv_bfloat16. Is only being read.

Returns

long long int
- h converted to a signed 64-bit integer.

Description

Convert the nv_bfloat16 floating-point value h to a signed 64-bit integer in round-up mode. NaN inputs return a long long int with hex value of 0x8000000000000000.

__host____device__ long long int __bfloat162ll_rz (const __nv_bfloat16 h)
Convert a nv_bfloat16 to a signed 64-bit integer in round-towards-zero mode.

Parameters

h
- nv_bfloat16. Is only being read.

Returns

long long int
- h converted to a signed 64-bit integer.

Description

Convert the nv_bfloat16 floating-point value h to a signed 64-bit integer in round-towards-zero mode. NaN inputs return a long long int with hex value of 0x8000000000000000.
__device__ short int __bfloat162short_rd (const __nv_bfloat16 h)
Convert a __nv_bfloat16 to a signed short integer in round-down mode.

Parameters
h
- __nv_bfloat16. Is only being read.

Returns
short int
  - h converted to a signed short integer.

Description
Convert the __nv_bfloat16 floating-point value h to a signed short integer in round-down mode. NaN inputs are converted to 0.

__device__ short int __bfloat162short_rn (const __nv_bfloat16 h)
Convert a __nv_bfloat16 to a signed short integer in round-to-nearest-even mode.

Parameters
h
- __nv_bfloat16. Is only being read.

Returns
short int
  - h converted to a signed short integer.

Description
Convert the __nv_bfloat16 floating-point value h to a signed short integer in round-to-nearest-even mode. NaN inputs are converted to 0.
__device__ short int __bfloat162short_ru (const __nv_bfloat16 h)

Convert a `nv_bfloat16` to a signed short integer in round-up mode.

**Parameters**

- h - `nv_bfloat16`. Is only being read.

**Returns**

- `short int` - h converted to a signed short integer.

**Description**

Convert the `nv_bfloat16` floating-point value h to a signed short integer in round-up mode. NaN inputs are converted to 0.

__host____device__ short int __bfloat162short_rz (const __nv_bfloat16 h)

Convert a `nv_bfloat16` to a signed short integer in round-towards-zero mode.

**Parameters**

- h - `nv_bfloat16`. Is only being read.

**Returns**

- `short int` - h converted to a signed short integer.

**Description**

Convert the `nv_bfloat16` floating-point value h to a signed short integer in round-towards-zero mode. NaN inputs are converted to 0.
__host__ __device__ unsigned char __bfloat162uchar_rz (const __nv_bfloat16 h)
Convert a nv_bfloat16 to an unsigned char in round-towards-zero mode.

Parameters
h
- nv_bfloat16. Is only being read.

Returns
unsigned char
  - h converted to an unsigned char.

Description
Convert the nv_bfloat16 floating-point value h to an unsigned char in round-towards-zero mode. NaN inputs are converted to 0.

__device__ unsigned int __bfloat162uint_rd (const __nv_bfloat16 h)
Convert a nv_bfloat16 to an unsigned integer in round-down mode.

Parameters
h
- nv_bfloat16. Is only being read.

Returns
unsigned int
  - h converted to an unsigned integer.

Description
Convert the nv_bfloat16 floating-point value h to an unsigned integer in round-down mode. NaN inputs are converted to 0.
__device__ unsigned int __bfloat162uint rn (const __nv_bfloat16 h)

Convert a nv_bfloat16 to an unsigned integer in round-to-nearest-even mode.

Parameters

h

- nv_bfloat16. Is only being read.

Returns

unsigned int

▸ h converted to an unsigned integer.

Description

Convert the nv_bfloat16 floating-point value h to an unsigned integer in round-to-nearest-even mode. NaN inputs are converted to 0.

__device__ unsigned int __bfloat162uint ru (const __nv_bfloat16 h)

Convert a nv_bfloat16 to an unsigned integer in round-up mode.

Parameters

h

- nv_bfloat16. Is only being read.

Returns

unsigned int

▸ h converted to an unsigned integer.

Description

Convert the nv_bfloat16 floating-point value h to an unsigned integer in round-up mode. NaN inputs are converted to 0.
__host__ __device__ unsigned int __bfloat162uint_rz (const __nv_bfloat16 h)
Convert a nv_bfloat16 to an unsigned integer in round-towards-zero mode.

Parameters
h
- nv_bfloat16. Is only being read.

Returns
unsigned int
▸ h converted to an unsigned integer.

Description
Convert the nv_bfloat16 floating-point value h to an unsigned integer in round-towards-zero mode. NaN inputs are converted to 0.

__device__ unsigned long long int __bfloat162ull_rd (const __nv_bfloat16 h)
Convert a nv_bfloat16 to an unsigned 64-bit integer in round-down mode.

Parameters
h
- nv_bfloat16. Is only being read.

Returns
unsigned long long int
▸ h converted to an unsigned 64-bit integer.

Description
Convert the nv_bfloat16 floating-point value h to an unsigned 64-bit integer in round-down mode. NaN inputs return 0x8000000000000000.
__device__ unsigned long long int __bfloat162ull_rn (const __nv_bfloat16 h)
Convert a nv_bfloat16 to an unsigned 64-bit integer in round-to-nearest-even mode.

Parameters
h
- nv_bfloat16. Is only being read.

Returns
unsigned long long int
▶ h converted to an unsigned 64-bit integer.

Description
Convert the nv_bfloat16 floating-point value h to an unsigned 64-bit integer in round-to-nearest-even mode. NaN inputs return 0x8000000000000000.

__device__ unsigned long long int __bfloat162ull_ru (const __nv_bfloat16 h)
Convert a nv_bfloat16 to an unsigned 64-bit integer in round-up mode.

Parameters
h
- nv_bfloat16. Is only being read.

Returns
unsigned long long int
▶ h converted to an unsigned 64-bit integer.

Description
Convert the nv_bfloat16 floating-point value h to an unsigned 64-bit integer in round-up mode. NaN inputs return 0x8000000000000000.
__host__ __device__ unsigned long long int __bfloat162ull_rz (const __nv_bfloat16 h)
Convert a nv_bfloat16 to an unsigned 64-bit integer in round-towards-zero mode.

Parameters

h
- nv_bfloat16. Is only being read.

Returns
unsigned long long int

- h converted to an unsigned 64-bit integer.

Description
Convert the nv_bfloat16 floating-point value h to an unsigned 64-bit integer in round-towards-zero mode. NaN inputs return 0x8000000000000000.

__device__ unsigned short int __bfloat162ushort_rd (const __nv_bfloat16 h)
Convert a nv_bfloat16 to an unsigned short integer in round-down mode.

Parameters

h
- nv_bfloat16. Is only being read.

Returns
unsigned short int

- h converted to an unsigned short integer.

Description
Convert the nv_bfloat16 floating-point value h to an unsigned short integer in round-down mode. NaN inputs are converted to 0.
__device__ unsigned short int __bfloat162ushort_rn (const
__nv_bfloat16 h)
Convert a nv_bfloat16 to an unsigned short integer in round-to-nearest-even mode.

Parameters
h
- nv_bfloat16. Is only being read.

Returns
unsigned short int
‣ h converted to an unsigned short integer.

Description
Convert the nv_bfloat16 floating-point value h to an unsigned short integer in round-to-nearest-even mode. NaN inputs are converted to 0.

__device__ unsigned short int __bfloat162ushort_ru (const
__nv_bfloat16 h)
Convert a nv_bfloat16 to an unsigned short integer in round-up mode.

Parameters
h
- nv_bfloat16. Is only being read.

Returns
unsigned short int
‣ h converted to an unsigned short integer.

Description
Convert the nv_bfloat16 floating-point value h to an unsigned short integer in round-up mode. NaN inputs are converted to 0.
__host__ __device__ unsigned short int __bfloat162ushort_rz (const __nv_bfloat16 h)
Convert a nv_bfloat16 to an unsigned short integer in round-towards-zero mode.

Parameters
h
- nv_bfloat16. Is only being read.

Returns
unsigned short int
- h converted to an unsigned short integer.

Description
Convert the nv_bfloat16 floating-point value h to an unsigned short integer in round-towards-zero mode. NaN inputs are converted to 0.

__host__ __device__ short int __bfloat16_as_short (const __nv_bfloat16 h)
Reinterprets bits in a nv_bfloat16 as a signed short integer.

Parameters
h
- nv_bfloat16. Is only being read.

Returns
short int
- The reinterpreted value.

Description
Reinterprets the bits in the nv_bfloat16 floating-point number h as a signed short integer.
__host__ __device__ unsigned short int __bfloat16_as_ushort (const __nv_bfloat16 h)
Reinterprets bits in a __nv_bfloat16 as an unsigned short integer.

Parameters
h
- __nv_bfloat16. Is only being read.

Returns
unsigned short int
- The reinterpreted value.

Description
Reinterprets the bits in the __nv_bfloat16 floating-point h as an unsigned short number.

__host__ __device__ double2bfloat16 (const double a)
Converts double number to __nv_bfloat16 precision in round-to-nearest-even mode and returns __nv_bfloat16 with converted value.

Parameters
a
- double. Is only being read.

Returns
__nv_bfloat16
- a converted to __nv_bfloat16.

Description
Converts double number a to __nv_bfloat16 precision in round-to-nearest-even mode.

__host__ __device__ float22bfloat162_rn (const float2 a)
Converts both components of float2 number to __nv_bfloat16 precision in round-to-nearest-even mode and returns __nv_bfloat162 with converted values.

Parameters
a
- float2. Is only being read.
Returns

nv_bfloat16

- The `nv_bfloat16` which has corresponding halves equal to the converted float2 components.

Description

Converts both components of float2 to `nv_bfloat16` precision in round-to-nearest-even mode and combines the results into one `nv_bfloat16` number. Low 16 bits of the return value correspond to `a.x` and high 16 bits of the return value correspond to `a.y`.

```c
__host__ __device__ float2bfloat16 (const float a)
```

Converts float number to `nv_bfloat16` precision in round-to-nearest-even mode and returns `nv_bfloat16` with converted value.

Parameters

- `a` - float. Is only being read.

Returns

nv_bfloat16

- `a` converted to `nv_bfloat16`.

Description

Converts float number `a` to `nv_bfloat16` precision in round-to-nearest-even mode.

```c
__host__ __device__ float2bffloat162_rn (const float a)
```

Converts input to `nv_bfloat16` precision in round-to-nearest-even mode and populates both halves of `nv_bfloat16` with converted value.

Parameters

- `a` - float. Is only being read.

Returns

nv_bfloat162

- The `nv_bfloat162` value with both halves equal to the converted `nv_bfloat16` precision number.
Description
Converts input \texttt{a} to \texttt{nv\_bfloate} precision in round-to-nearest-even mode and populates both halves of \texttt{nv\_bfloate} with converted value.

\texttt{__host\_\_device\_\_float2bfloate\_rd (const float a)}
Converts float number to \texttt{nv\_bfloate} precision in round-down mode and returns \texttt{nv\_bfloate} with converted value.

Parameters
\texttt{a}
- float. Is only being read.

Returns
\texttt{nv\_bfloate}
\begin{itemize}
  \item \texttt{a} converted to \texttt{nv\_bfloate}.
\end{itemize}

Description
Converts float number \texttt{a} to \texttt{nv\_bfloate} precision in round-down mode.

\texttt{__host\_\_device\_\_float2bfloate\_rn (const float a)}
Converts float number to \texttt{nv\_bfloate} precision in round-to-nearest-even mode and returns \texttt{nv\_bfloate} with converted value.

Parameters
\texttt{a}
- float. Is only being read.

Returns
\texttt{nv\_bfloate}
\begin{itemize}
  \item \texttt{a} converted to \texttt{nv\_bfloate}.
\end{itemize}

Description
Converts float number \texttt{a} to \texttt{nv\_bfloate} precision in round-to-nearest-even mode.
__host____device____float2bfloat16_ru (const float a)
Converts float number to nv_bfloat16 precision in round-up mode and returns nv_bfloat16 with converted value.

Parameters
a
- float. Is only being read.

Returns
nv_bfloat16
▶ a converted to nv_bfloat16.

Description
Converts float number a to nv_bfloat16 precision in round-up mode.

__host____device____float2bfloat16_rz (const float a)
Converts float number to nv_bfloat16 precision in round-towards-zero mode and returns nv_bfloat16 with converted value.

Parameters
a
- float. Is only being read.

Returns
nv_bfloat16
▶ a converted to nv_bfloat16.

Description
Converts float number a to nv_bfloat16 precision in round-towards-zero mode.

__host____device____floats2bfloat162_rn (const float a, const float b)
Converts both input floats to nv_bfloat16 precision in round-to-nearest-even mode and returns nv_bfloat162 with converted values.

Parameters
a
- float. Is only being read.
Returns

\texttt{nv\_bf\_float162}

- The \texttt{nv\_bf\_float162} value with corresponding halves equal to the converted input floats.

Description

Converts both input floats to \texttt{nv\_bf\_float16} precision in round-to-nearest-even mode and combines the results into one \texttt{nv\_bf\_float162} number. Low 16 bits of the return value correspond to the input \(a\), high 16 bits correspond to the input \(b\).

\texttt{__host\_\_device\_\_halves2bf\_float162 (const __nv\_bf\_float16 a, const __nv\_bf\_float16 b)}

Combines two \texttt{nv\_bf\_float16} numbers into one \texttt{nv\_bf\_float162} number.

Parameters

\textbf{a}

- \texttt{nv\_bf\_float16}. Is only being read.

\textbf{b}

- \texttt{nv\_bf\_float16}. Is only being read.

Returns

\texttt{nv\_bf\_float162}

- The \texttt{nv\_bf\_float162} with one \texttt{nv\_bf\_float16} equal to \(a\) and the other to \(b\).

Description

Combines two input \texttt{nv\_bf\_float16} number \(a\) and \(b\) into one \texttt{nv\_bf\_float162} number. Input \(a\) is stored in low 16 bits of the return value, input \(b\) is stored in high 16 bits of the return value.

\texttt{__host\_\_device\_\_high2bf\_float16 (const __nv\_bf\_float162 a)}

Returns high 16 bits of \texttt{nv\_bf\_float162} input.

Parameters

\textbf{a}

- \texttt{nv\_bf\_float162}. Is only being read.
Returns
nv_bfloat16

- The high 16 bits of the input.

Description
Returns high 16 bits of nv_bfloat16 input a.

__host__ __device__ high2bfloat162 (const __nv_bfloat162 a)
Extracts high 16 bits from nv_bfloat162 input.

Parameters
a
- nv_bfloat162. Is only being read.

Returns
nv_bfloat162

- The nv_bfloat162 with both halves equal to the high 16 bits of the input.

Description
Extracts high 16 bits from nv_bfloat162 input a and returns a new nv_bfloat162 number which has both halves equal to the extracted bits.

__host__ __device__ float __high2float (const __nv_bfloat162 a)
Converts high 16 bits of nv_bfloat162 to float and returns the result.

Parameters
a
- nv_bfloat162. Is only being read.

Returns
float

- The high 16 bits of a converted to float.
Description
Converts high 16 bits of `nv_bfloat162` input `a` to 32-bit floating-point number and returns the result.

```c
__host__ __device__ highs2bfloat162 (const __nv_bfloat162 a, const __nv_bfloat162 b)
```
Extracts high 16 bits from each of the two `nv_bfloat162` inputs and combines into one `nv_bfloat162` number.

Parameters
- `a` - `nv_bfloat162`. Is only being read.
- `b` - `nv_bfloat162`. Is only being read.

Returns
- `nv_bfloat162` - The high 16 bits of `a` and of `b`.

Description
Extracts high 16 bits from each of the two `nv_bfloat162` inputs and combines into one `nv_bfloat162` number. High 16 bits from input `a` is stored in low 16 bits of the return value, high 16 bits from input `b` is stored in high 16 bits of the return value.

```c
__device__ int2bfloat16_rd (const int i)
```
Convert a signed integer to a `nv_bfloat16` in round-down mode.

Parameters
- `i` - `int`. Is only being read.

Returns
- `nv_bfloat16` - `i` converted to `nv_bfloat16`.

Description
Convert the signed integer value `i` to a `nv_bfloat16` floating-point value in round-down mode.
\_\_host\_\_\_device\_\_\_int2bfloat16\_rn (const int i)
Convert a signed integer to a nv\_bfself16 in round-to-nearest-even mode.

Parameters

\texttt{i}
- int. Is only being read.

Returns

nv\_bfself16
\texttt{i} converted to nv\_bfself16.

Description

Convert the signed integer value \texttt{i} to a nv\_bfself16 floating-point value in round-to-nearest-even mode.

\_\_device\_\_\_int2bfloat16\_ru (const int i)
Convert a signed integer to a nv\_bfself16 in round-up mode.

Parameters

\texttt{i}
- int. Is only being read.

Returns

nv\_bfself16
\texttt{i} converted to nv\_bfself16.

Description

Convert the signed integer value \texttt{i} to a nv\_bfself16 floating-point value in round-up mode.

\_\_device\_\_\_int2bfloat16\_rz (const int i)
Convert a signed integer to a nv\_bfself16 in round-towards-zero mode.

Parameters

\texttt{i}
- int. Is only being read.

Returns

nv\_bfself16
- \( i \) converted to `nv_bfloat16`.

**Description**

Convert the signed integer value \( i \) to a `nv_bfloat16` floating-point value in round-towards-zero mode.

```c
__device__ __ldca (const __nv_bfloat16 *ptr)
```

Generates a `ld.global.ca` load instruction.

**Parameters**

- **ptr**
  - memory location

**Returns**

The value pointed by `ptr`

```c
__device__ __ldca (const __nv_bfloat162 *ptr)
```

Generates a `ld.global.ca` load instruction.

**Parameters**

- **ptr**
  - memory location

**Returns**

The value pointed by `ptr`

```c
__device__ __ldcg (const __nv_bfloat16 *ptr)
```

Generates a `ld.global.cg` load instruction.

**Parameters**

- **ptr**
  - memory location

**Returns**

The value pointed by `ptr`
__device__ ldcg (const __nv_bfloat16 *ptr)
Generates a `ld.global.cg` load instruction.

Parameters
ptr
- memory location

Returns
The value pointed by `ptr`

__device__ ldcs (const __nv_bfloat16 *ptr)
Generates a `ld.global.cs` load instruction.

Parameters
ptr
- memory location

Returns
The value pointed by `ptr`

__device__ ldcs (const __nv_bfloat16 *ptr)
Generates a `ld.global.cs` load instruction.

Parameters
ptr
- memory location

Returns
The value pointed by `ptr`

__device__ ldcv (const __nv_bfloat16 *ptr)
Generates a `ld.global.cv` load instruction.

Parameters
ptr
- memory location

Returns
The value pointed by `ptr`
__device____ldcv (const __nv_bfloat16 *ptr)
Generates a `ld.global.cv` load instruction.

Parameters
ptr
- memory location

Returns
The value pointed by `ptr`

__device____ldg (const __nv_bfloat16 *ptr)
Generates a `ld.global.nc` load instruction.

Parameters
ptr
- memory location

Returns
The value pointed by `ptr`

__device____ldg (const __nv_bfloat162 *ptr)
Generates a `ld.global.nc` load instruction.

Parameters
ptr
- memory location

Returns
The value pointed by `ptr`

__device____ldlu (const __nv_bfloat16 *ptr)
Generates a `ld.global.lu` load instruction.

Parameters
ptr
- memory location

Returns
The value pointed by `ptr`
__device____ldlu (const __nv_bfloat162 *ptr)
Generates a `ld.global.lu` load instruction.

Parameters
ptr
- memory location

Returns
The value pointed by `ptr`

__device____ll2bfloat16_rd (const long long int i)
Convert a signed 64-bit integer to a nv_bfloat16 in round-down mode.

Parameters
i
- long long int. Is only being read.

Returns
nv_bfloat16
► i converted to nv_bfloat16.

Description
Convert the signed 64-bit integer value i to a nv_bfloat16 floating-point value in round-down mode.

__host____device____ll2bfloat16_rn (const long long int i)
Convert a signed 64-bit integer to a nv_bfloat16 in round-to-nearest-even mode.

Parameters
i
- long long int. Is only being read.

Returns
nv_bfloat16
► i converted to nv_bfloat16.

Description
Convert the signed 64-bit integer value i to a nv_bfloat16 floating-point value in round-to-nearest-even mode.
__device__ ll2bfloat16_ru (const long long int i)
Convert a signed 64-bit integer to a nv_bfloat16 in round-up mode.

Parameters

i
  - long long int. Is only being read.

Returns

nv_bfloat16
  - i converted to nv_bfloat16.

Description

Convert the signed 64-bit integer value \(i\) to a nv_bfloat16 floating-point value in round-up mode.

__device__ ll2bfloat16_rz (const long long int i)
Convert a signed 64-bit integer to a nv_bfloat16 in round-towards-zero mode.

Parameters

i
  - long long int. Is only being read.

Returns

nv_bfloat16
  - i converted to nv_bfloat16.

Description

Convert the signed 64-bit integer value \(i\) to a nv_bfloat16 floating-point value in round-towards-zero mode.

__host__ __device__ low2bfloat16 (const __nv_bfloat162 a)
Returns low 16 bits of nv_bfloat162 input.

Parameters

a
  - nv_bfloat162. Is only being read.
Returns
nv_bfloat16

Returns \texttt{nv\_bfloat16} which contains low 16 bits of the input \texttt{a}.

Description
Returns low 16 bits of \texttt{nv\_bfloat16} input \texttt{a}.

\texttt{\_host\_\_device\_\_low2bfloa162 (const \_\_nv\_bfloat162 \texttt{a})}
Extracts low 16 bits from \texttt{nv\_bfloat162} input.

Parameters
\texttt{a}
- \texttt{nv\_bfloat162}. Is only being read.

Returns
\texttt{nv\_bfloat162}

The \texttt{nv\_bfloat162} with both halves equal to the low 16 bits of the input.

Description
Extracts low 16 bits from \texttt{nv\_bfloat162} input \texttt{a} and returns a new \texttt{nv\_bfloat162} number which has both halves equal to the extracted bits.

\texttt{\_host\_\_device\_\_float\_\_low2float (const \_\_nv\_bfloat162 \texttt{a})}
Converts low 16 bits of \texttt{nv\_bfloat162} to float and returns the result.

Parameters
\texttt{a}
- \texttt{nv\_bfloat162}. Is only being read.

Returns
\texttt{float}

The low 16 bits of \texttt{a} converted to float.
Description
Converts low 16 bits of \texttt{nv\_bfloat162} input \texttt{a} to 32-bit floating-point number and returns the result.

\texttt{\_\_host\_\_device\_\_lowhigh2highlow (const \_\_nv\_bfloat162 a)}
Swaps both halves of the \texttt{nv\_bfloat162} input.

Parameters
\texttt{a} - \texttt{nv\_bfloat162}. Is only being read.

Returns
\texttt{nv\_bfloat162}

- \texttt{a} with its halves being swapped.

Description
Swaps both halves of the \texttt{nv\_bfloat162} input and returns a new \texttt{nv\_bfloat162} number with swapped halves.

\texttt{\_\_host\_\_device\_\_lows2bfloat162 (const \_\_nv\_bfloat162 a, const \_\_nv\_bfloat162 b)}
Extracts low 16 bits from each of the two \texttt{nv\_bfloat162} inputs and combines into one \texttt{nv\_bfloat162} number.

Parameters
\texttt{a}
- \texttt{nv\_bfloat162}. Is only being read.

\texttt{b}
- \texttt{nv\_bfloat162}. Is only being read.

Returns
\texttt{nv\_bfloat162}

- The low 16 bits of \texttt{a} and of \texttt{b}. 
Description

Extracts low 16 bits from each of the two `nv_bfloat162` inputs and combines into one `nv_bfloat162` number. Low 16 bits from input `a` is stored in low 16 bits of the return value, low 16 bits from input `b` is stored in high 16 bits of the return value.

```c
__device__ shfl_down_sync (const unsigned mask, const __nv_bfloat16 var, const unsigned int delta, const int width)
```

Exchange a variable between threads within a warp. Copy from a thread with higher ID relative to the caller.

Parameters

- **mask**
  - unsigned int. Is only being read.
- **var**
  - `nv_bfloat16`. Is only being read.
- **delta**
  - int. Is only being read.
- **width**
  - int. Is only being read.

Returns

Returns the 2-byte word referenced by `var` from the source thread ID as `nv_bfloat16`. If the source thread ID is out of range or the source thread has exited, the calling thread’s own `var` is returned.

Description

Calculates a source thread ID by adding `delta` to the caller’s thread ID. The value of `var` held by the resulting thread ID is returned; this has the effect of shifting `var` down the warp by `delta` threads. If `width` is less than `warpSize` then each subsection of the warp behaves as a separate entity with a starting logical thread ID of 0. As for `__shfl_up_sync()`, the ID number of the source thread will not wrap around the value of `width` and so the upper `delta` threads will remain unchanged.

Note:

For more details for this function see the Warp Shuffle Functions section in the CUDA C++ Programming Guide.
__device__ shfl_down_sync (const unsigned mask, const __nv_bfloat16 var, const unsigned int delta, const int width)
Exchange a variable between threads within a warp. Copy from a thread with higher ID relative to the caller.

Parameters

mask
- unsigned int. Is only being read.

var
- nv_bfloat16. Is only being read.

delta
- int. Is only being read.

width
- int. Is only being read.

Returns

Returns the 4-byte word referenced by var from the source thread ID as nv_bfloat16. If the source thread ID is out of range or the source thread has exited, the calling thread’s own var is returned.

Description

Calculates a source thread ID by adding delta to the caller’s thread ID. The value of var held by the resulting thread ID is returned: this has the effect of shifting var down the warp by delta threads. If width is less than warpSize then each subsection of the warp behaves as a separate entity with a starting logical thread ID of 0. As for __shfl_up_sync[], the ID number of the source thread will not wrap around the value of width and so the upper delta threads will remain unchanged.

Note:
For more details for this function see the Warp Shuffle Functions section in the CUDA C++ Programming Guide.

__device__ shfl_sync (const unsigned mask, const __nv_bfloat16 var, const int delta, const int width)
Exchange a variable between threads within a warp. Direct copy from indexed thread.

Parameters

mask
- unsigned int. Is only being read.
__device__ __shfl_sync (const unsigned mask, const __nv_bfloat16 var, const int delta, const int width)

Returns the 2-byte word referenced by var from the source thread ID as __nv_bfloat16. If the source thread ID is out of range or the source thread has exited, the calling thread’s own var is returned.

Description

Returns the value of var held by the thread whose ID is given by delta. If width is less than warpSize then each subsection of the warp behaves as a separate entity with a starting logical thread ID of 0. If delta is outside the range [0:width-1], the value returned corresponds to the value of var held by the delta modulo width (i.e. within the same subsection). width must have a value which is a power of 2; results are undefined if width is not a power of 2, or is a number greater than warpSize.

Note:
For more details for this function see the Warp Shuffle Functions section in the CUDA C++ Programming Guide.

Parameters

**mask**
- unsigned int. Is only being read.

**var**
- __nv_bfloat16. Is only being read.

**delta**
- int. Is only being read.

**width**
- int. Is only being read.
**Returns**
Returns the 4-byte word referenced by var from the source thread ID as \texttt{nv\_bfloat16}. If the source thread ID is out of range or the source thread has exited, the calling thread’s own var is returned.

**Description**
Returns the value of var held by the thread whose ID is given by delta. If width is less than warpSize then each subsection of the warp behaves as a separate entity with a starting logical thread ID of 0. If delta is outside the range \([0:width-1]\), the value returned corresponds to the value of var held by the delta modulo width (i.e. within the same subsection). width must have a value which is a power of 2; results are undefined if width is not a power of 2, or is a number greater than warpSize.

**Note:**
For more details for this function see the Warp Shuffle Functions section in the CUDA C++ Programming Guide.

**__device__**\_shfl\_up\_sync (const unsigned mask, const \_nv\_bfloat16 var, const unsigned int delta, const int width)
Exchange a variable between threads within a warp. Copy from a thread with lower ID relative to the caller.

**Parameters**
- **mask** - unsigned int. Is only being read.
- **var** - \texttt{nv\_bfloat16}. Is only being read.
- **delta** - int. Is only being read.
- **width** - int. Is only being read.

**Returns**
Returns the 2-byte word referenced by var from the source thread ID as \texttt{nv\_bfloat16}. If the source thread ID is out of range or the source thread has exited, the calling thread’s own var is returned.
Description
Calculates a source thread ID by subtracting delta from the caller's lane ID. The value of var held by the resulting lane ID is returned: in effect, var is shifted up the warp by delta threads. If width is less than warpSize then each subsection of the warp behaves as a separate entity with a starting logical thread ID of 0. The source thread index will not wrap around the value of width, so effectively the lower delta threads will be unchanged. width must have a value which is a power of 2; results are undefined if width is not a power of 2, or is a number greater than warpSize.

Note:
For more details for this function see the Warp Shuffle Functions section in the CUDA C++ Programming Guide.

__device__ __shfl_up_sync (const unsigned mask, const __nv_bfloat162 var, const unsigned int delta, const int width)
Exchange a variable between threads within a warp. Copy from a thread with lower ID relative to the caller.

Parameters
- **mask**
  - unsigned int. Is only being read.
- **var**
  - __nv_bfloat162. Is only being read.
- **delta**
  - int. Is only being read.
- **width**
  - int. Is only being read.

Returns
Returns the 4-byte word referenced by var from the source thread ID as __nv_bfloat162. If the source thread ID is out of range or the source thread has exited, the calling thread's own var is returned.

Description
Calculates a source thread ID by subtracting delta from the caller's lane ID. The value of var held by the resulting lane ID is returned: in effect, var is shifted up the warp by delta threads. If width is less than warpSize then each subsection of the warp behaves as a separate entity with a starting logical thread ID of 0. The source thread index will not wrap around the value of width, so effectively the lower delta threads will be unchanged. width must have a value which
is a power of 2; results are undefined if width is not a power of 2, or is a number greater than warpSize.

Note:
For more details for this function see the Warp Shuffle Functions section in the CUDA C++ Programming Guide.

__device__ shfl_xor_sync (const unsigned mask, const __nv_bfloat16 var, const int delta, const int width)
Exchange a variable between threads within a warp. Copy from a thread based on bitwise XOR of own thread ID.

Parameters
mask
- unsigned int. Is only being read.
var
- __nv_bfloat16. Is only being read.
delta
- int. Is only being read.
width
- int. Is only being read.

Returns
Returns the 2-byte word referenced by var from the source thread ID as __nv_bfloat16. If the source thread ID is out of range or the source thread has exited, the calling thread's own var is returned.

Description
Calculates a source thread ID by performing a bitwise XOR of the caller's thread ID with mask: the value of var held by the resulting thread ID is returned. If width is less than warpSize then each group of width consecutive threads are able to access elements from earlier groups of threads, however if they attempt to access elements from later groups of threads their own value of var will be returned. This mode implements a butterfly addressing pattern such as is used in tree reduction and broadcast.

Note:
For more details for this function see the Warp Shuffle Functions section in the CUDA C++ Programming Guide.
__device__ shfl_xor_sync (const unsigned mask, const __nv_bfloat162 var, const int delta, const int width)

Exchange a variable between threads within a warp. Copy from a thread based on bitwise XOR of own thread ID.

Parameters

mask
- unsigned int. Is only being read.

var
- __nv_bfloat162. Is only being read.

delta
- int. Is only being read.

width
- int. Is only being read.

Returns

Returns the 4-byte word referenced by var from the source thread ID as __nv_bfloat162. If the source thread ID is out of range or the source thread has exited, the calling thread’s own var is returned.

Description

Calculates a source thread ID by performing a bitwise XOR of the caller’s thread ID with mask: the value of var held by the resulting thread ID is returned. If width is less than warpSize then each group of width consecutive threads are able to access elements from earlier groups of threads, however if they attempt to access elements from later groups of threads their own value of var will be returned. This mode implements a butterfly addressing pattern such as is used in tree reduction and broadcast.

Note:
For more details for this function see the Warp Shuffle Functions section in the CUDA C++ Programming Guide.

__device__ short2bfloat16_rd (const short int i)

Convert a signed short integer to a __nv_bfloat16 in round-down mode.

Parameters

i
- short int. Is only being read.
Returns
nv_bfloat16

- i converted to nv_bfloat16.

Description
Convert the signed short integer value i to a nv_bfloat16 floating-point value in round-down mode.

__host__ __device__ __short2bfloat16_rn (const short int i)
Convert a signed short integer to a nv_bfloat16 in round-to-nearest-even mode.

Parameters
i
- short int. Is only being read.

Returns
nv_bfloat16

- i converted to nv_bfloat16.

Description
Convert the signed short integer value i to a nv_bfloat16 floating-point value in round-to-nearest-even mode.

__device__ __short2bfloat16_ru (const short int i)
Convert a signed short integer to a nv_bfloat16 in round-up mode.

Parameters
i
- short int. Is only being read.

Returns
nv_bfloat16

- i converted to nv_bfloat16.

Description
Convert the signed short integer value i to a nv_bfloat16 floating-point value in round-up mode.
__device__ __short2bfloat16_rz (const short int i)
Convert a signed short integer to a nv_bfloat16 in round-towards-zero mode.

Parameters
i
- short int. Is only being read.

Returns
nv_bfloat16
- i converted to nv_bfloat16.

Description
Convert the signed short integer value i to a nv_bfloat16 floating-point value in round-towards-zero mode.

__host__ __device__ __short_as_bfloat16 (const short int i)
Reinterprets bits in a signed short integer as a nv_bfloat16.

Parameters
i
- short int. Is only being read.

Returns
nv_bfloat16
- The reinterpreted value.

Description
Reinterprets the bits in the signed short integer i as a nv_bfloat16 floating-point number.

__device__ void __stcg (const __nv_bfloat16 *ptr, const __nv_bfloat16 value)
Generates a `st.global.cg` store instruction.

Parameters
ptr
- memory location
value
- the value to be stored
__device__ void __stcg (const __nv_bfloat162 *ptr, const __nv_bfloat162 value)
Generates a `st.global.cg` store instruction.

Parameters
ptr
- memory location

value
- the value to be stored

__device__ void __stcs (const __nv_bfloat16 *ptr, const __nv_bfloat16 value)
Generates a `st.global.cs` store instruction.

Parameters
ptr
- memory location

value
- the value to be stored

__device__ void __stcs (const __nv_bfloat162 *ptr, const __nv_bfloat162 value)
Generates a `st.global.cs` store instruction.

Parameters
ptr
- memory location

value
- the value to be stored

__device__ void __stwb (const __nv_bfloat16 *ptr, const __nv_bfloat16 value)
Generates a `st.global.wb` store instruction.

Parameters
ptr
- memory location

value
- the value to be stored
__device__ void __stwb (const __nv_bfloat162 *ptr, const __nv_bfloat162 value)
Generates a `st.global.wb` store instruction.

Parameters

ptr     - memory location
value   - the value to be stored

__device__ void __stwt (const __nv_bfloat16 *ptr, const __nv_bfloat16 value)
Generates a `st.global.wt` store instruction.

Parameters

ptr     - memory location
value   - the value to be stored

__device__ void __stwt (const __nv_bfloat162 *ptr, const __nv_bfloat162 value)
Generates a `st.global.wt` store instruction.

Parameters

ptr     - memory location
value   - the value to be stored

__device__ uint2bf16_rd (const unsigned int i)
Convert an unsigned integer to a nv_bfloat16 in round-down mode.

Parameters

i       - unsigned int. Is only being read.

Returns

nv_bfloat16
i converted to nv_bfloat16.

Description

Convert the unsigned integer value \( i \) to a nv_bfloat16 floating-point value in round-down mode.

\[
_{\text{__host____device____}}\text{uint2bfloat16}_\text{rn} \ (\text{const unsigned int \( i \))}
\]

Convert an unsigned integer to a nv_bfloat16 in round-to-nearest-even mode.

Parameters

\( i \)

- unsigned int. Is only being read.

Returns

nv_bfloat16

\( i \) converted to nv_bfloat16.

Description

Convert the unsigned integer value \( i \) to a nv_bfloat16 floating-point value in round-to-nearest-even mode.

\[
_{\text{__device____}}\text{uint2bfloat16}_\text{ru} \ (\text{const unsigned int \( i \))}
\]

Convert an unsigned integer to a nv_bfloat16 in round-up mode.

Parameters

\( i \)

- unsigned int. Is only being read.

Returns

nv_bfloat16

\( i \) converted to nv_bfloat16.

Description

Convert the unsigned integer value \( i \) to a nv_bfloat16 floating-point value in round-up mode.
**__device__** `uint2bfloat16_rz (const unsigned int i)`
Convert an unsigned integer to a `nv_bfloat16` in round-towards-zero mode.

**Parameters**

`i` - unsigned int. Is only being read.

**Returns**

`nv_bfloat16`
- `i` converted to `nv_bfloat16`.

**Description**

Convert the unsigned integer value `i` to a `nv_bfloat16` floating-point value in round-towards-zero mode.

**__device__** `ull2bfloat16_rd (const unsigned long long int i)`
Convert an unsigned 64-bit integer to a `nv_bfloat16` in round-down mode.

**Parameters**

`i` - unsigned long long int. Is only being read.

**Returns**

`nv_bfloat16`
- `i` converted to `nv_bfloat16`.

**Description**

Convert the unsigned 64-bit integer value `i` to a `nv_bfloat16` floating-point value in round-down mode.

**__host__** `__device__` `ull2bfloat16_rn (const unsigned long long int i)`
Convert an unsigned 64-bit integer to a `nv_bfloat16` in round-to-nearest-even mode.

**Parameters**

`i` - unsigned long long int. Is only being read.
Returns
nv_bfloat16

- `i` converted to `nv_bfloat16`.

Description
Convert the unsigned 64-bit integer value `i` to a `nv_bfloat16` floating-point value in round-to-nearest-even mode.

__device__ull2bfloat16_ru (const unsigned long long int `i`)
Convert an unsigned 64-bit integer to a `nv_bfloat16` in round-up mode.

Parameters
- `i` - unsigned long long int. Is only being read.

Returns
nv_bfloat16

- `i` converted to `nv_bfloat16`.

Description
Convert the unsigned 64-bit integer value `i` to a `nv_bfloat16` floating-point value in round-up mode.

__device__ull2bfloat16_rz (const unsigned long long int `i`)
Convert an unsigned 64-bit integer to a `nv_bfloat16` in round-towards-zero mode.

Parameters
- `i` - unsigned long long int. Is only being read.

Returns
nv_bfloat16

- `i` converted to `nv_bfloat16`.

Description
Convert the unsigned 64-bit integer value `i` to a `nv_bfloat16` floating-point value in round-towards-zero mode.
__device____ushort2bfloat16_rd (const unsigned short int i)
Convert an unsigned short integer to a nv_bfloat16 in round-down mode.

Parameters
i
  - unsigned short int. Is only being read.

Returns
nv_bfloat16
  ➤ i converted to nv_bfloat16.

Description
Convert the unsigned short integer value i to a nv_bfloat16 floating-point value in round-down mode.

__host____device____ushort2bfloat16_rn (const unsigned short int i)
Convert an unsigned short integer to a nv_bfloat16 in round-to-nearest-even mode.

Parameters
i
  - unsigned short int. Is only being read.

Returns
nv_bfloat16
  ➤ i converted to nv_bfloat16.

Description
Convert the unsigned short integer value i to a nv_bfloat16 floating-point value in round-to-nearest-even mode.

__device____ushort2bfloat16_ru (const unsigned short int i)
Convert an unsigned short integer to a nv_bfloat16 in round-up mode.

Parameters
i
  - unsigned short int. Is only being read.
Returns
nv_bfloat16

- i converted to nv_bfloat16.

Description
Convert the unsigned short integer value i to a nv_bfloat16 floating-point value in round-up mode.

__device__ ushort2bfloat16_rz (const unsigned short int i)
Convert an unsigned short integer to a nv_bfloat16 in round-towards-zero mode.

Parameters
i
- unsigned short int. Is only being read.

Returns
nv_bfloat16

- i converted to nv_bfloat16.

Description
Convert the unsigned short integer value i to a nv_bfloat16 floating-point value in round-towards-zero mode.

__host__ __device__ ushort_as_bfloat16 (const unsigned short int i)
Reinterprets bits in an unsigned short integer as a nv_bfloat16.

Parameters
i
- unsigned short int. Is only being read.

Returns
nv_bfloat16

- The reinterpreted value.

Description
Reinterprets the bits in the unsigned short integer i as a nv_bfloat16 floating-point number.
__host__ __device__ make_bfloat162 (const __nv_bfloat16 x, const __nv_bfloat16 y)
Vector function, combines two nv_bfloat16 numbers into one nv_bfloat162 number.

Parameters
x
   - nv_bfloat16. Is only being read.
y
   - nv_bfloat16. Is only being read.

Returns
__nv_bfloat162
   The __nv_bfloat162 vector with one half equal to x and the other to y.

Description
Combines two input nv_bfloat16 number x and y into one nv_bfloat162 number. Input x is stored in low 16 bits of the return value, input y is stored in high 16 bits of the return value.

1.3.7. Bfloat16 Math Functions

Bfloat16 Precision Intrinsics
To use these functions, include the header file cuda_bf16.h in your program.

__device__ hceil (const __nv_bfloat16 h)
Calculate ceiling of the input argument.

Parameters
h
   - nv_bfloat16. Is only being read.

Returns
nv_bfloat16
   The smallest integer value not less than h.

Description
Compute the smallest integer value not less than h.
__device__hcos (const __nv_bfloat16 a)
Calculates nv_bfloat16 cosine in round-to-nearest-even mode.

Parameters

a
- nv_bfloat16. Is only being read.

Returns

nv_bfloat16
- The cosine of a.

Description

Calculates nv_bfloat16 cosine of input a in round-to-nearest-even mode.

NOTE: this function’s implementation calls cosf(float) function and is exposed to compiler optimizations. Specifically, --use_fast_math flag changes cosf(float) into an intrinsic __cosf(float), which has less accurate numeric behavior.

__device__hexp (const __nv_bfloat16 a)
Calculates nv_bfloat16 natural exponential function in round-to-nearest-even mode.

Parameters

a
- nv_bfloat16. Is only being read.

Returns

nv_bfloat16
- The natural exponential function on a.

Description

Calculates nv_bfloat16 natural exponential function of input a in round-to-nearest-even mode.

__device__hexp10 (const __nv_bfloat16 a)
Calculates nv_bfloat16 decimal exponential function in round-to-nearest-even mode.

Parameters

a
- nv_bfloat16. Is only being read.
Returns
nv_bfloat16
➤ The decimal exponential function on a.

Description
Calculates nv_bfloat16 decimal exponential function of input a in round-to-nearest-even mode.

__device__hexp2 (const __nv_bfloat16 a)
Calculates nv_bfloat16 binary exponential function in round-to-nearest-even mode.

Parameters
a
- nv_bfloat16. Is only being read.

Returns
nv_bfloat16
➤ The binary exponential function on a.

Description
Calculates nv_bfloat16 binary exponential function of input a in round-to-nearest-even mode.

__device__hfloor (const __nv_bfloat16 h)
Calculate the largest integer less than or equal to h.

Parameters
h
- nv_bfloat16. Is only being read.

Returns
nv_bfloat16
➤ The largest integer value which is less than or equal to h.

Description
Calculate the largest integer value which is less than or equal to h.
__device__ hlog (const __nv_bfloat16 a)
Calculates \texttt{nv\_bfloa}t16 natural logarithm in round-to-nearest-even mode.

Parameters
a
- \texttt{nv\_bfloa}t16. Is only being read.

Returns
\texttt{nv\_bfloa}t16

\>
- The natural logarithm of \texttt{a}.

Description
Calculates \texttt{nv\_bfloa}t16 natural logarithm of input \texttt{a} in round-to-nearest-even mode.

__device__ hlog10 (const __nv_bfloat16 a)
Calculates \texttt{nv\_bfloa}t16 decimal logarithm in round-to-nearest-even mode.

Parameters
a
- \texttt{nv\_bfloa}t16. Is only being read.

Returns
\texttt{nv\_bfloa}t16

\>
- The decimal logarithm of \texttt{a}.

Description
Calculates \texttt{nv\_bfloa}t16 decimal logarithm of input \texttt{a} in round-to-nearest-even mode.

__device__ hlog2 (const __nv_bfloat16 a)
Calculates \texttt{nv\_bfloa}t16 binary logarithm in round-to-nearest-even mode.

Parameters
a
- \texttt{nv\_bfloa}t16. Is only being read.

Returns
\texttt{nv\_bfloa}t16

\>
- The binary logarithm of \texttt{a}.
**__device__hrcp (const __nv_bfloat16 a)**

Calculates \( \text{nv\_bfloat16} \) reciprocal in round-to-nearest-even mode.

**Parameters**

- **a**
  - \( \text{nv\_bfloat16} \). Is only being read.

**Returns**

- \( \text{nv\_bfloat16} \)
  - The reciprocal of \( a \).

**__device__hrint (const __nv_bfloat16 h)**

Round input to nearest integer value in \( \text{nv\_bfloat16} \) floating-point number.

**Parameters**

- **h**
  - \( \text{nv\_bfloat16} \). Is only being read.

**Returns**

- \( \text{nv\_bfloat16} \)
  - The nearest integer to \( h \).

**Description**

Round \( h \) to the nearest integer value in \( \text{nv\_bfloat16} \) floating-point format, with \( \text{bfloat16way} \) cases rounded to the nearest even integer value.
\texttt{__device\_hsqrt (const \_\_nv\_bfloat16 a)}

Calculates \texttt{nv\_bfloat16 reciprocal square root} in round-to-nearest-even mode.

**Parameters**

\texttt{a}

- \texttt{nv\_bfloat16}. Is only being read.

**Returns**

\texttt{nv\_bfloat16}

- The reciprocal square root of \texttt{a}.

**Description**

Calculates \texttt{nv\_bfloat16 reciprocal square root} of input \texttt{a} in round-to-nearest-even mode.

\texttt{__device\_hsin (const \_\_nv\_bfloat16 a)}

Calculates \texttt{nv\_bfloat16 sine} in round-to-nearest-even mode.

**Parameters**

\texttt{a}

- \texttt{nv\_bfloat16}. Is only being read.

**Returns**

\texttt{nv\_bfloat16}

- The sine of \texttt{a}.

**Description**

Calculates \texttt{nv\_bfloat16 sine} of input \texttt{a} in round-to-nearest-even mode.

\texttt{NOTE: this function's implementation calls sinf(float) function and is exposed to compiler optimizations. Specifically, --use\_fast\_math flag changes sinf(float) into an intrinsic __sinf(float), which has less accurate numeric behavior.}

\texttt{__device\_hsqrt (const \_\_nv\_bfloat16 a)}

Calculates \texttt{nv\_bfloat16 square root} in round-to-nearest-even mode.

**Parameters**

\texttt{a}

- \texttt{nv\_bfloat16}. Is only being read.
Returns
nv_bfloat16

- The square root of a.

Description
Calculates \( \text{nv\_bfloat16} \) square root of input \( a \) in round-to-nearest-even mode.

\_device\_htrunc (const \_nv\_bfloat16 \( h \))
Truncate input argument to the integral part.

Parameters
\( h \)
- \( \text{nv\_bfloat16} \). Is only being read.

Returns
nv_bfloat16

- The truncated integer value.

Description
Round \( h \) to the nearest integer value that does not exceed \( h \) in magnitude.

1.3.8. Bfloat162 Math Functions

Bfloat16 Precision Intrinsics
To use these functions, include the header file \texttt{cuda\_bf16.h} in your program.

\_device\_h2ceil (const \_nv\_bfloat162 \( h \))
Calculate \( \text{nv\_bfloat162} \) vector ceiling of the input argument.

Parameters
\( h \)
- \( \text{nv\_bfloat162} \). Is only being read.

Returns
nv_bfloat162

- The vector of smallest integers not less than \( h \).
Description
For each component of vector \( h \) compute the smallest integer value not less than \( h \).

```c
__device__ h2cos (const __nv_bfloat162 a)
```
Calculates \( \text{nv\_bfloat162} \) vector cosine in round-to-nearest-even mode.

Parameters
- \( a \) - \( \text{nv\_bfloat162} \). Is only being read.

Returns
- \( \text{nv\_bfloat162} \)

The elementwise cosine on vector \( a \).

Description
Calculates \( \text{nv\_bfloat162} \) cosine of input vector \( a \) in round-to-nearest-even mode.

NOTE: this function’s implementation calls `cosf(float)` function and is exposed to compiler optimizations. Specifically, `--use_fast_math` flag changes `cosf(float)` into an intrinsic `__cosf(float)`, which has less accurate numeric behavior.

```c
__device__ h2exp (const __nv_bfloat162 a)
```
Calculates \( \text{nv\_bfloat162} \) vector exponential function in round-to-nearest-even mode.

Parameters
- \( a \) - \( \text{nv\_bfloat162} \). Is only being read.

Returns
- \( \text{nv\_bfloat162} \)

The elementwise exponential function on vector \( a \).

Description
Calculates \( \text{nv\_bfloat162} \) exponential function of input vector \( a \) in round-to-nearest-even mode.
__device__ h2exp10 (const __nv_bfloat162 a)
Calculates __nv_bfloat162 vector decimal exponential function in round-to-nearest-even mode.

Parameters
a
- __nv_bfloat162. Is only being read.

Returns
__nv_bfloat162
- The elementwise decimal exponential function on vector a.

Description
Calculates __nv_bfloat162 decimal exponential function of input vector a in round-to-nearest-even mode.

__device__ h2exp2 (const __nv_bfloat162 a)
Calculates __nv_bfloat162 vector binary exponential function in round-to-nearest-even mode.

Parameters
a
- __nv_bfloat162. Is only being read.

Returns
__nv_bfloat162
- The elementwise binary exponential function on vector a.

Description
Calculates __nv_bfloat162 binary exponential function of input vector a in round-to-nearest-even mode.

__device__ h2floor (const __nv_bfloat162 h)
Calculate the largest integer less than or equal to h.

Parameters
h
- __nv_bfloat162. Is only being read.
Returns
nv_bfloat162
  ▶ The vector of largest integers which is less than or equal to \( h \).

Description
For each component of vector \( h \) calculate the largest integer value which is less than or equal to \( h \).

\_\_device\_\_h2log (const \_\_nv\_bfloat162 a)
Calculates \_\_nv\_bfloat162 vector natural logarithm in round-to-nearest-even mode.

Parameters
a
  - nv_bfloat162. Is only being read.

Returns
nv_bfloat162
  ▶ The elementwise natural logarithm on vector \( a \).

Description
Calculates \_\_nv\_bfloat162 natural logarithm of input vector \( a \) in round-to-nearest-even mode.

\_\_device\_\_h2log10 (const \_\_nv\_bfloat162 a)
Calculates \_\_nv\_bfloat162 vector decimal logarithm in round-to-nearest-even mode.

Parameters
a
  - nv_bfloat162. Is only being read.

Returns
nv_bfloat162
  ▶ The elementwise decimal logarithm on vector \( a \).

Description
Calculates \_\_nv\_bfloat162 decimal logarithm of input vector \( a \) in round-to-nearest-even mode.
**__device__ h2log2 (const __nv_bfloat162 a)**
Calculates \( \text{nv\_bfloat162} \) vector binary logarithm in round-to-nearest-even mode.

**Parameters**

\( a \)
- \( \text{nv\_bfloat162} \). Is only being read.

**Returns**

\( \text{nv\_bfloat162} \)
- The elementwise binary logarithm on vector \( a \).

**Description**

Calculates \( \text{nv\_bfloat162} \) binary logarithm of input vector \( a \) in round-to-nearest-even mode.

**__device__ h2rcp (const __nv_bfloat162 a)**
Calculates \( \text{nv\_bfloat162} \) vector reciprocal in round-to-nearest-even mode.

**Parameters**

\( a \)
- \( \text{nv\_bfloat162} \). Is only being read.

**Returns**

\( \text{nv\_bfloat162} \)
- The elementwise reciprocal on vector \( a \).

**Description**

Calculates \( \text{nv\_bfloat162} \) reciprocal of input vector \( a \) in round-to-nearest-even mode.

**__device__ h2rint (const __nv_bfloat162 h)**
Round input to nearest integer value in \( \text{nv\_bfloat16} \) floating-point number.

**Parameters**

\( h \)
- \( \text{nv\_bfloat16} \). Is only being read.

**Returns**

\( \text{nv\_bfloat16} \)
- The vector of rounded integer values.
Description
Round each component of \( \text{nv\_bfloat162} \) vector \( h \) to the nearest integer value in \( \text{nv\_bfloat16} \) floating-point format, with bfloat16way cases rounded to the nearest even integer value.

\[
\text{__device\_h2rsqrt (const __nv\_bfloat162 a)}
\]
Calculates \( \text{nv\_bfloat162} \) vector reciprocal square root in round-to-nearest-even mode.

Parameters
\( a \)
- \( \text{nv\_bfloat162} \). Is only being read.

Returns
\( \text{nv\_bfloat162} \)
- The elementwise reciprocal square root on vector \( a \).

Description
Calculates \( \text{nv\_bfloat162} \) reciprocal square root of input vector \( a \) in round-to-nearest-even mode.

\[
\text{__device\_h2sin (const __nv\_bfloat162 a)}
\]
Calculates \( \text{nv\_bfloat162} \) vector sine in round-to-nearest-even mode.

Parameters
\( a \)
- \( \text{nv\_bfloat162} \). Is only being read.

Returns
\( \text{nv\_bfloat162} \)
- The elementwise sine on vector \( a \).

Description
Calculates \( \text{nv\_bfloat162} \) sine of input vector \( a \) in round-to-nearest-even mode.

NOTE: this function’s implementation calls \( \text{sinf\_float} \) function and is exposed to compiler optimizations. Specifically, \(--\text{use\_fast\_math} \) flag changes \( \text{sinf\_float} \) into an intrinsic \( \text{__sinf\_float} \), which has less accurate numeric behavior.
__device__ h2sqrt (const __nv_bfloat162 a)
Calculates nv_bfloat162 vector square root in round-to-nearest-even mode.

Parameters
a
- nv_bfloat162. Is only being read.

Returns
nv_bfloat162
▷ The elementwise square root on vector a.

Description
Calculates nv_bfloat162 square root of input vector a in round-to-nearest-even mode.

__device__ h2trunc (const __nv_bfloat162 h)
Truncate nv_bfloat162 vector input argument to the integral part.

Parameters
h
- nv_bfloat162. Is only being read.

Returns
nv_bfloat162
▷ The truncated h.

Description
Round each component of vector h to the nearest integer value that does not exceed h in magnitude.

1.4. Mathematical Functions

CUDA mathematical functions are always available in device code.

Host implementations of the common mathematical functions are mapped in a platform-specific way to standard math library functions, provided by the host compiler and respective host libm where available. Some functions, not available with the host compilers, are implemented in crt/math_functions.hpp header file. For example, see erfinv(). Other, less common functions, like rhypot(), cyl_bessel_i0() are only available in device code.
Note that many floating-point and integer functions names are overloaded for different argument types. For example, the `log()` function has the following prototypes:

```
‡ double log(double x);
 float log(float x);
 float logf(float x);
```

Note also that due to implementation constraints, certain math functions from std::namespace may be callable in device code even via explicitly qualified std::names. However, such use is discouraged, since this capability is unsupported, unverified, undocumented, not portable, and may change without notice.

1.5. Single Precision Mathematical Functions

This section describes single precision mathematical functions. To use these functions you do not need to include any additional header files in your program.

__device__ float acosf(float x)

Calculate the arc cosine of the input argument.

Returns

Result will be in radians, in the interval [0, \pi] for x inside [-1, +1].

- `acosf(1)` returns +0.
- `acosf(x)` returns NaN for x outside [-1, +1].

Description

Calculate the principal value of the arc cosine of the input argument x.

Note:

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float acoshf(float x)

Calculate the nonnegative inverse hyperbolic cosine of the input argument.

Returns

Result will be in the interval [0, +\infty].
- \( \text{acosh}(1) \) returns 0.
- \( \text{acosh}(x) \) returns NaN for \( x \) in the interval \([-\infty, 1)\).
- \( \text{acosh}(+\infty) \) returns \( +\infty \).

**Description**

Calculate the nonnegative inverse hyperbolic cosine of the input argument \( x \).

**Note:**

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

```c
__device__ float asinf(float x)
```

**Calculate the arc sine of the input argument.**

**Returns**

Result will be in radians, in the interval \([-\pi/2, +\pi/2]\) for \( x \) inside \([-1, +1]\).

- \( \text{asinf}(\pm 0) \) returns \( \pm 0 \).
- \( \text{asinf}(x) \) returns NaN for \( x \) outside \([-1, +1]\).

**Description**

Calculate the principal value of the arc sine of the input argument \( x \).

**Note:**

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

```c
__device__ float asinhf(float x)
```

**Calculate the inverse hyperbolic sine of the input argument.**

**Returns**

- \( \text{asinhf}(\pm 0) \) returns \( \pm 0 \).
- \( \text{asinhf}(\pm \infty) \) returns \( \pm \infty \).

**Description**

Calculate the inverse hyperbolic sine of the input argument \( x \).
__device__ float atan2f(float y, float x)
Calculate the arc tangent of the ratio of first and second input arguments.

Returns
Result will be in radians, in the interval \([-\pi, +\pi]\).

- atan2f(±0, -0) returns ±\pi.
- atan2f(±0, +0) returns ±0.
- atan2f(±0, x) returns ±\pi for x < 0.
- atan2f(±0, x) returns ±0 for x > 0.
- atan2f(y, ±0) returns −\pi/2 for y < 0.
- atan2f(y, ±0) returns \pi/2 for y > 0.
- atan2f(±y, −∞) returns ±\pi for finite y > 0.
- atan2f(±y, +∞) returns ±0 for finite y > 0.
- atan2f(±∞, x) returns ±\pi/2 for finite x.
- atan2f(±∞, −∞) returns ±3\pi/4.
- atan2f(±∞, +∞) returns ±\pi/4.

Description
Calculate the principal value of the arc tangent of the ratio of first and second input arguments \(y / x\). The quadrant of the result is determined by the signs of inputs \(y\) and \(x\).

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
\textbf{__device__ float atanf(float x)}

Calculate the arc tangent of the input argument.

\textbf{Returns}

Result will be in radians, in the interval \[-\pi/2, +\pi/2\].

- \texttt{atanf( 0 )} returns 0.
- \texttt{atanf( \pm \infty \)} returns \pm \pi/2.

\textbf{Description}

Calculate the principal value of the arc tangent of the input argument \(x\).

\textbf{Note:}

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

\textbf{__device__ float atanhf(float x)}

Calculate the inverse hyperbolic tangent of the input argument.

\textbf{Returns}

- \texttt{atanhf( 0 )} returns 0.
- \texttt{atanhf( \pm 1 \)} returns \pm \infty.
- \texttt{atanhf(x)} returns NaN for \(x\) outside interval \([-1, 1]\).

\textbf{Description}

Calculate the inverse hyperbolic tangent of the input argument \(x\).

\textbf{Note:}

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
__device__ float cbrtf (float x)
Calculate the cube root of the input argument.

Returns
Returns \( x^{1/3} \).

- \( cbrtf( \pm 0 ) \) returns \( \pm 0 \).
- \( cbrtf( \pm \infty ) \) returns \( \pm \infty \).

Description
Calculate the cube root of \( x \), \( x^{1/3} \).

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float ceilf (float x)
Calculate ceiling of the input argument.

Returns
Returns \( [x] \) expressed as a floating-point number.

- \( ceilf( \pm 0 ) \) returns \( \pm 0 \).
- \( ceilf( \pm \infty ) \) returns \( \pm \infty \).

Description
Compute the smallest integer value not less than \( x \).

__device__ float copysignf (float x, float y)
Create value with given magnitude, copying sign of second value.

Returns
Returns a value with the magnitude of \( x \) and the sign of \( y \).

Description
Create a floating-point value with the magnitude \( x \) and the sign of \( y \).
__device__ float cosf(float x)
Calculate the cosine of the input argument.

Returns

‣ cosf(±0) returns 1.
‣ cosf(±∞) returns NaN.

Description
Calculate the cosine of the input argument \(x\) (measured in radians).

Note:

‣ For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
‣ This function is affected by the --use_fast_math compiler flag. See the CUDA C++ Programming Guide, Mathematical Functions Appendix, Intrinsic Functions section for a complete list of functions affected.

__device__ float coshf(float x)
Calculate the hyperbolic cosine of the input argument.

Returns

‣ coshf(±0) returns 1.
‣ coshf(±∞) returns +∞.

Description
Calculate the hyperbolic cosine of the input argument \(x\).

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
__device__ float cospif (float x)

Calculate the cosine of the input argument \( x \pi \).

Returns

- \( \text{cospif}( \pm 0 ) \) returns 1.
- \( \text{cospif}( \pm \infty ) \) returns NaN.

Description

Calculate the cosine of \( x \pi \) (measured in radians), where \( x \) is the input argument.

Note:

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float cyl_bessel_i0f (float x)

Calculate the value of the regular modified cylindrical Bessel function of order 0 for the input argument.

Returns

Returns the value of the regular modified cylindrical Bessel function of order 0.

Description

Calculate the value of the regular modified cylindrical Bessel function of order 0 for the input argument \( x \), \( I_0(x) \).

Note:

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float cyl_bessel_i1f (float x)

Calculate the value of the regular modified cylindrical Bessel function of order 1 for the input argument.

Returns

Returns the value of the regular modified cylindrical Bessel function of order 1.
**Description**

Calculate the value of the regular modified cylindrical Bessel function of order 1 for the input argument $x$, $I_1(x)$.

**Note:**

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float erfcf (float x)

Calculate the complementary error function of the input argument.

**Returns**

- $\text{erfcf}(-\infty)$ returns 2.
- $\text{erfcf}(+\infty)$ returns $+0$.

**Description**

Calculate the complementary error function of the input argument $x$, $1 - \text{erf}(x)$.

**Note:**

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float erfcinvf (float x)

Calculate the inverse complementary error function of the input argument.

**Returns**

- $\text{erfcinvf}(\pm 0)$ returns $+\infty$.
- $\text{erfcinvf}(2)$ returns $-\infty$.
- $\text{erfcinvf}(x)$ returns NaN for $x$ outside $[0, 2]$.

**Description**

Calculate the inverse complementary error function $\text{erfc}^{-1}(x)$, of the input argument $x$ in the interval $[0, 2]$. 
__device__ float erfcxf (float x)
Calculate the scaled complementary error function of the input argument.

Returns
- \( \text{erfcxf}( -\infty ) \) returns \( +\infty \).
- \( \text{erfcxf}( +\infty ) \) returns \( +0 \).

Description
Calculate the scaled complementary error function of the input argument \( x \), \( e^{x^2} \cdot \text{erfc}(x) \).

__device__ float erff (float x)
Calculate the error function of the input argument.

Returns
- \( \text{erff}( \pm 0 ) \) returns \( \pm 0 \).
- \( \text{erff}( \pm \infty ) \) returns \( \pm 1 \).

Description
Calculate the value of the error function for the input argument \( x \),
\[
\frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} \, dt.
\]
__device__ float erfinvf (float x)
Calculate the inverse error function of the input argument.

Returns
▷ erfinvf(±0) returns ±0.
▷ erfinvf(1) returns +∞.
▷ erfinvf(-1) returns −∞.
▷ erfinvf(x) returns NaN for x outside [-1, +1].

Description
Calculate the inverse error function $\text{erf}^{-1}(x)$, of the input argument $x$ in the interval [-1, 1].

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float exp10f (float x)
Calculate the base 10 exponential of the input argument.

Returns
▷ exp10f(±0) returns 1.
▷ exp10f(−∞) returns +0.
▷ exp10f(+∞) returns +∞.

Description
Calculate $10^x$, the base 10 exponential of the input argument $x$.

Note:
▷ For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
▷ This function is affected by the --use_fast_math compiler flag. See the CUDA C++ Programming Guide, Mathematical Functions Appendix, Intrinsic Functions section for a complete list of functions affected.
__device__ float exp2f (float x)
Calculate the base 2 exponential of the input argument.

Returns
- $\exp(\pm 0)$ returns 1.
- $\exp(-\infty)$ returns +0.
- $\exp(+\infty)$ returns $+\infty$.

Description
Calculate $2^x$, the base 2 exponential of the input argument $x$.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float expf (float x)
Calculate the base $e$ exponential of the input argument.

Returns
- $\exp(\pm 0)$ returns 1.
- $\exp(-\infty)$ returns +0.
- $\exp(+\infty)$ returns $+\infty$.

Description
Calculate $e^x$, the base $e$ exponential of the input argument $x$.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
- This function is affected by the $--use\_fast\_math$ compiler flag. See the CUDA C++ Programming Guide, Mathematical Functions Appendix, Intrinsic Functions section for a complete list of functions affected.
**__device__ float expm1f (float x)***

Calculate the base $e$ exponential of the input argument, minus 1.

**Returns**

- `expm1f(±0)` returns `±0`.
- `expm1f(−∞)` returns `-1`.
- `expm1f(+∞)` returns `+∞`.

**Description**

Calculate $e^x - 1$, the base $e$ exponential of the input argument $x$, minus 1.

**Note:**

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

**__device__ float fabsf (float x)***

Calculate the absolute value of its argument.

**Returns**

Returns the absolute value of its argument.

- `fabsf(±∞)` returns `+∞`.
- `fabsf(±0)` returns `+0`.
- `fabsf(NaN)` returns an unspecified NaN.

**Description**

Calculate the absolute value of the input argument $x$.

**Note:**

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
__device__ float fdimf (float x, float y)

Compute the positive difference between x and y.

Returns

Returns the positive difference between x and y.

- fdimf(x, y) returns x - y if x > y.
- fdimf(x, y) returns +0 if x \leq y.

Description

Compute the positive difference between x and y. The positive difference is x - y when x > y and +0 otherwise.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float fdividef (float x, float y)

Divide two floating-point values.

Returns

Returns x / y.

Description

Compute x divided by y. If --use_fast_math is specified, use __fdividef() for higher performance, otherwise use normal division.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
- This function is affected by the --use_fast_math compiler flag. See the CUDA C++ Programming Guide, Mathematical Functions Appendix, Intrinsic Functions section for a complete list of functions affected.
__device__ float floorf (float x)
Calculate the largest integer less than or equal to x.

Returns
Returns |x| expressed as a floating-point number.

- floorf( ±∞ ) returns ±∞.
- floorf( ±0 ) returns ±0.

Description
Calculate the largest integer value which is less than or equal to x.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float fmaf (float x, float y, float z)
Compute x × y + z as a single operation.

Returns
Returns the rounded value of x × y + z as a single operation.

- fmaf( ±∞ , ±0 , z) returns NaN.
- fmaf( ±0 , ±∞ , z) returns NaN.
- fmaf(x, y, −∞ ) returns NaN if x × y is an exact +∞.
- fmaf(x, y, +∞ ) returns NaN if x × y is an exact −∞.

Description
Compute the value of x × y + z as a single ternary operation. After computing the value to infinite precision, the value is rounded once.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
__device__ float fmaxf (float x, float y)
Determine the maximum numeric value of the arguments.

Returns
Returns the maximum numeric values of the arguments \( x \) and \( y \).

- If both arguments are NaN, returns NaN.
- If one argument is NaN, returns the numeric argument.

Description
Determines the maximum numeric value of the arguments \( x \) and \( y \). Treats NaN arguments as missing data. If one argument is a NaN and the other is legitimate numeric value, the numeric value is chosen.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float fminf (float x, float y)
Determine the minimum numeric value of the arguments.

Returns
Returns the minimum numeric value of the arguments \( x \) and \( y \).

- If both arguments are NaN, returns NaN.
- If one argument is NaN, returns the numeric argument.

Description
Determines the minimum numeric value of the arguments \( x \) and \( y \). Treats NaN arguments as missing data. If one argument is a NaN and the other is legitimate numeric value, the numeric value is chosen.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
__device__ float fmodf (float x, float y)

Calculate the floating-point remainder of $x / y$.

**Returns**

- Returns the floating-point remainder of $x / y$.
- $\text{fmodf}(\pm 0, y)$ returns $\pm 0$ if $y$ is not zero.
- $\text{fmodf}(x, \pm \infty)$ returns $x$ if $x$ is finite.
- $\text{fmodf}(x, y)$ returns NaN if $x$ is $\pm \infty$ or $y$ is zero.
- If either argument is NaN, NaN is returned.

**Description**

Calculate the floating-point remainder of $x / y$. The floating-point remainder of the division operation $x / y$ calculated by this function is exactly the value $x - n*y$, where $n$ is $x / y$ with its fractional part truncated. The computed value will have the same sign as $x$, and its magnitude will be less than the magnitude of $y$.

**Note:**
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float frexp (float x, int *nptr)

Extract mantissa and exponent of a floating-point value.

**Returns**

Returns the fractional component $m$.

- $\text{frexp}(\pm 0, \text{nptr})$ returns $\pm 0$ and stores zero in the location pointed to by nptr.
- $\text{frexp}(\pm \infty, \text{nptr})$ returns $\pm \infty$ and stores an unspecified value in the location to which nptr points.
- $\text{frexp}(\text{NaN}, y)$ returns a NaN and stores an unspecified value in the location to which nptr points.

**Description**

Decomposes the floating-point value $x$ into a component $m$ for the normalized fraction element and another term $n$ for the exponent. The absolute value of $m$ will be greater than or equal to
0.5 and less than 1.0 or it will be equal to 0; \( x = m \cdot 2^n \). The integer exponent \( n \) will be stored in the location to which \( nptr \) points.

**Note:**
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

```c
__device__ float hypotf (float x, float y)
```
Calculate the square root of the sum of squares of two arguments.

**Returns**
Returns the length of the hypotenuse \( \sqrt{x^2 + y^2} \).

- \( \text{hypotf}(x, y) \), \( \text{hypotf}(y, x) \), and \( \text{hypotf}(x, -y) \) are equivalent.
- \( \text{hypotf}(x, \pm 0) \) is equivalent to \( \text{fabsf}(x) \).
- \( \text{hypotf}(\pm \infty, y) \) returns \( +\infty \), even if \( y \) is a NaN.

**Description**
Calculates the length of the hypotenuse of a right triangle whose two sides have lengths \( x \) and \( y \) without undue overflow or underflow.

**Note:**
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

```c
__device__ int ilogbf (float x)
```
Compute the unbiased integer exponent of the argument.

**Returns**
- If successful, returns the unbiased exponent of the argument.
- \( \text{ilogbf}(\pm 0) \) returns \( \text{INT\_MIN} \).
- \( \text{ilogbf}(\text{NaN}) \) returns \( \text{INT\_MIN} \).
- \( \text{ilogbf}(\pm \infty) \) returns \( \text{INT\_MAX} \).
- Note: above behavior does not take into account \( \text{FP\_ILOGB0} \) nor \( \text{FP\_ILOGBNAN} \).
Description
Calculates the unbiased integer exponent of the input argument x.

**Note:**
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

```__device__ __RETURN_TYPE isfinite (float a)`
Determine whether argument is finite.

**Returns**

- With Visual Studio 2013 host compiler: `__RETURN_TYPE` is 'bool'. Returns true if and only if `a` is a finite value.
- With other host compilers: `__RETURN_TYPE` is 'int'. Returns a nonzero value if and only if `a` is a finite value.

Description
Determine whether the floating-point value `a` is a finite value (zero, subnormal, or normal and not infinity or NaN).

```__device__ __RETURN_TYPE isinf (float a)`
Determine whether argument is infinite.

**Returns**

- With Visual Studio 2013 host compiler: `__RETURN_TYPE` is 'bool'. Returns true if and only if `a` is an infinite value.
- With other host compilers: `__RETURN_TYPE` is 'int'. Returns a nonzero value if and only if `a` is an infinite value.

Description
Determine whether the floating-point value `a` is an infinite value (positive or negative).
__device__ __RETURN_TYPE isnan (float a)
Determine whether argument is a NaN.

Returns
▶ With Visual Studio 2013 host compiler: __RETURN_TYPE is ‘bool’. Returns true if and only if a is a NaN value.
▶ With other host compilers: __RETURN_TYPE is ‘int’. Returns a nonzero value if and only if a is a NaN value.

Description
Determine whether the floating-point value \( a \) is a NaN.

__device__ float j0f (float x)
Calculate the value of the Bessel function of the first kind of order 0 for the input argument.

Returns
Returns the value of the Bessel function of the first kind of order 0.
▶ \( j_0(x) \) returns +0.
▶ \( j_0(\text{NaN}) \) returns NaN.

Description
Calculate the value of the Bessel function of the first kind of order 0 for the input argument \( x \), \( J_0(x) \).

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float j1f (float x)
Calculate the value of the Bessel function of the first kind of order 1 for the input argument.

Returns
Returns the value of the Bessel function of the first kind of order 1.
▶ \( j_1(x) \) returns \( -0 \).
▶ \( j_1(\text{NaN}) \) returns \( -0 \).
j1f(NaN) returns NaN.

Description
Calculate the value of the Bessel function of the first kind of order 1 for the input argument \( x, J_1(x) \).

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float jnf (int n, float x)
Calculate the value of the Bessel function of the first kind of order \( n \) for the input argument.

Returns
Returns the value of the Bessel function of the first kind of order \( n \).
- jnf(n, NaN) returns NaN.
- jnf(n, x) returns NaN for \( n < 0 \).
- jnf(n, +\infty) returns +0.

Description
Calculate the value of the Bessel function of the first kind of order \( n \) for the input argument \( x, J_n(x) \).

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float ldexpf (float x, int exp)
Calculate the value of \( x \cdot 2^{\exp} \).

Returns
- ldexpf(x, exp) is equivalent to scalbnf(x, exp).

Description
Calculate the value of \( x \cdot 2^{\exp} \) of the input arguments x and exp.
CUDA Math API v12.4

__device__ float lgammaf (float x)
Calculate the natural logarithm of the absolute value of the gamma function of the input argument.

Returns
- lgammaf(1) returns +0.
- lgammaf(2) returns +0.
- lgammaf(x) returns +∞ if x ≤ 0 and x is an integer.
- lgammaf(−∞) returns +∞.
- lgammaf(+∞) returns +∞.

Description
Calculate the natural logarithm of the absolute value of the gamma function of the input argument x, namely the value of \[ \log_e \left( \int_0^\infty e^{-t} t^{x-1} dt \right) \]

__device__ long long int llrintf (float x)
Round input to nearest integer value.

Returns
Returns rounded integer value.

Description
Round x to the nearest integer value, with halfway cases rounded to the nearest even integer value. If the result is outside the range of the return type, the behavior is undefined.
__device__ long long int llroundf(float x)
Round to nearest integer value.

Returns
Returns rounded integer value.

Description
Round x to the nearest integer value, with halfway cases rounded away from zero. If the result is outside the range of the return type, the behavior is undefined.

Note:
This function may be slower than alternate rounding methods. See `llrintf`.

__device__ float log10f (float x)
Calculate the base 10 logarithm of the input argument.

Returns
- \( \log_{10}(\pm 0) \) returns \(-\infty\).
- \( \log_{10}(1) \) returns \(+0\).
- \( \log_{10}(x) \) returns NaN for \( x < 0 \).
- \( \log_{10}(+\infty) \) returns \(+\infty\).

Description
Calculate the base 10 logarithm of the input argument \( x \).

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
- This function is affected by the `--use_fast_math` compiler flag. See the CUDA C++ Programming Guide, Mathematical Functions Appendix, Intrinsic Functions section for a complete list of functions affected.
__device__ float log1pf(float x)
Calculate the value of $\log_e(1 + x)$.

Returns

- $\log1pf(\pm 0)$ returns $\pm 0$.
- $\log1pf(-1)$ returns $-\infty$.
- $\log1pf(x)$ returns NaN for $x < -1$.
- $\log1pf(+\infty)$ returns $+\infty$.

Description

Calculate the value of $\log_e(1 + x)$ of the input argument $x$.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float log2f(float x)
Calculate the base 2 logarithm of the input argument.

Returns

- $\log2f(\pm 0)$ returns $-\infty$.
- $\log2f(1)$ returns $+0$.
- $\log2f(x)$ returns NaN for $x < 0$.
- $\log2f(+\infty)$ returns $+\infty$.

Description

Calculate the base 2 logarithm of the input argument $x$.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
This function is affected by the `--use_fast_math` compiler flag. See the CUDA C++ Programming Guide, Mathematical Functions Appendix, Intrinsic Functions section for a complete list of functions affected.

__device__ float logbf (float x)

Calculate the floating-point representation of the exponent of the input argument.

Returns

- `logbf(±0)` returns $-\infty$.
- `logbf(±\infty)` returns $+\infty$.

Description

Calculate the floating-point representation of the exponent of the input argument $x$.

Note:

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float logf (float x)

Calculate the natural logarithm of the input argument.

Returns

- `logf(±0)` returns $-\infty$.
- `logf(1)` returns $+0$.
- `logf(x)` returns NaN for $x < 0$.
- `logf(+\infty)` returns $+\infty$.

Description

Calculate the natural logarithm of the input argument $x$.

Note:

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
This function is affected by the --use_fast_math compiler flag. See the CUDA C++ Programming Guide, Mathematical Functions Appendix, Intrinsic Functions section for a complete list of functions affected.

__device__ long int lrintf (float x)
Round input to nearest integer value.

Returns
Returns rounded integer value.

Description
Round $x$ to the nearest integer value, with halfway cases rounded to the nearest even integer value. If the result is outside the range of the return type, the behavior is undefined.

__device__ long int lroundf (float x)
Round to nearest integer value.

Returns
Returns rounded integer value.

Description
Round $x$ to the nearest integer value, with halfway cases rounded away from zero. If the result is outside the range of the return type, the behavior is undefined.

Note:
This function may be slower than alternate rounding methods. See lrintf().

__device__ float max (const float a, const float b)
Calculate the maximum value of the input float arguments.

Description
Calculate the maximum value of the arguments $a$ and $b$. Behavior is equivalent to fmaxf() function.

Note, this is different from std:: specification
__device__ float min (const float a, const float b)
Calculate the minimum value of the input float arguments.

Description
Calculate the minimum value of the arguments a and b. Behavior is equivalent to fminf function.
Note, this is different from std:: specification

__device__ float modff (float x, float *iptr)
Break down the input argument into fractional and integral parts.

Returns
- modff(±x, iptr) returns a result with the same sign as x.
- modff(±∞, iptr) returns ±0 and stores ±∞ in the object pointed to by iptr.
- modff(NaN, iptr) stores a NaN in the object pointed to by iptr and returns a NaN.

Description
Break down the argument x into fractional and integral parts. The integral part is stored in the argument iptr. Fractional and integral parts are given the same sign as the argument x.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float nanf (const char *tagp)
Returns “Not a Number” value.

Returns
- nanf(tagp) returns NaN.

Description
Return a representation of a quiet NaN. Argument tagp selects one of the possible representations.
**__device__ float nearbyintf (float x)***

Round the input argument to the nearest integer.

**Returns**

- `nearbyintf(±0)` returns ±0.
- `nearbyintf(±∞)` returns ±∞.

**Description**

Round argument `x` to an integer value in single precision floating-point format. Uses round to nearest rounding, with ties rounding to even.

**__device__ float nextafterf (float x, float y)***

Return next representable single-precision floating-point value after argument `x` in the direction of `y`.

**Returns**

- `nextafterf(x, y) = y` if `x` equals `y`.
- `nextafterf(x, y) = NaN` if either `x` or `y` are `NaN`.

**Description**

Calculate the next representable single-precision floating-point value following `x` in the direction of `y`. For example, if `y` is greater than `x`, `nextafterf[]` returns the smallest representable number greater than `x`
__device__ float norm3df (float a, float b, float c)
Calculate the square root of the sum of squares of three coordinates of the argument.

Returns
Returns the length of the 3D vector $\sqrt{a^2 + b^2 + c^2}$.
- In the presence of an exactly infinite coordinate $+\infty$ is returned, even if there are NaNs.

Description
Calculates the length of three dimensional vector in Euclidean space without undue overflow or underflow.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float norm4df (float a, float b, float c, float d)
Calculate the square root of the sum of squares of four coordinates of the argument.

Returns
Returns the length of the 4D vector $\sqrt{a^2 + b^2 + c^2 + d^2}$.
- In the presence of an exactly infinite coordinate $+\infty$ is returned, even if there are NaNs.

Description
Calculates the length of four dimensional vector in Euclidean space without undue overflow or underflow.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
__device__ float normcdff (float x)

Calculate the standard normal cumulative distribution function.

Returns

- normcdff( +∞ ) returns 1.
- normcdff( −∞ ) returns +0

Description

Calculate the cumulative distribution function of the standard normal distribution for input argument x, \( \Phi(x) \).

Note:

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float normcdfinvf (float x)

Calculate the inverse of the standard normal cumulative distribution function.

Returns

- normcdfinvf( ±0 ) returns −∞.
- normcdfinvf(1) returns +∞.
- normcdfinvf(x) returns NaN if x is not in the interval [0,1].

Description

Calculate the inverse of the standard normal cumulative distribution function for input argument x, \( \Phi^{-1}(x) \). The function is defined for input values in the interval \((0, 1)\).

Note:

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
__device__ float normf (int dim, const float *p)
Calculate the square root of the sum of squares of any number of coordinates.

Returns
Returns the length of the dim-D vector $\sqrt{p_0^2 + p_1^2 + \ldots + p_{dim-1}^2}$.

- In the presence of an exactly infinite coordinate $+\infty$ is returned, even if there are NaNs.

Description
Calculates the length of a vector p, dimension of which is passed as an argument without undue overflow or underflow.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float powf (float x, float y)
Calculate the value of first argument to the power of second argument.

Returns
- $\text{powf}(\pm 0, y)$ returns $+\infty$ for $y$ an odd integer less than 0.
- $\text{powf}(\pm 0, y)$ returns $+\infty$ for $y$ less than 0 and not an odd integer.
- $\text{powf}(\pm 0, y)$ returns $0$ for $y$ an odd integer greater than 0.
- $\text{powf}(\pm 0, y)$ returns $0$ for $y > 0$ and not an odd integer.
- $\text{powf}(-1, \pm \infty)$ returns $1$.
- $\text{powf}(+1, y)$ returns $1$ for any $y$, even a NaN.
- $\text{powf}(x, \pm 0)$ returns $1$ for any $x$, even a NaN.
- $\text{powf}(x, y)$ returns a NaN for finite $x < 0$ and finite non-integer $y$.
- $\text{powf}(x, -\infty)$ returns $+\infty$ for $|x| < 1$.
- $\text{powf}(x, -\infty)$ returns $+\infty$ for $|x| > 1$.
- $\text{powf}(x, +\infty)$ returns $0$ for $|x| < 1$.
- $\text{powf}(x, +\infty)$ returns $+\infty$ for $|x| > 1$.
- $\text{powf}( -\infty, y)$ returns $0$ for $y$ an odd integer less than 0.
- powf(−∞, y) returns +0 for y < 0 and not an odd integer.
- powf(−∞, y) returns −∞ for y an odd integer greater than 0.
- powf(−∞, y) returns +∞ for y > 0 and not an odd integer.
- powf(+∞, y) returns +0 for y < 0.
- powf(+∞, y) returns +∞ for y > 0.

**Description**
Calculate the value of x to the power of y.

**Note:**
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
- This function is affected by the --use_fast_math compiler flag. See the CUDA C++ Programming Guide, Mathematical Functions Appendix, Intrinsic Functions section for a complete list of functions affected.

```device__ float rcbtf(float x)
```
Calculate reciprocal cube root function.

**Returns**
- rcbtf(±0) returns ±∞.
- rcbtf(±∞) returns ±0.

**Description**
Calculate reciprocal cube root function of x.

**Note:**
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
__device__ float remainderf (float x, float y)
Compute single-precision floating-point remainder.

Returns

- remainderf(x, ±0) returns NaN.
- remainderf(±∞, y) returns NaN.
- remainderf(x, ±∞) returns x for finite x.

Description
Computes single-precision floating-point remainder r of dividing x by y for nonzero y. Thus
\[ r = x - ny. \]
The value n is the integer value nearest \( \frac{x}{y} \). In the case when \( |n - \frac{x}{y}| = \frac{1}{2} \), the even n value is chosen.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float remquof (float x, float y, int *quo)
Compute single-precision floating-point remainder and part of quotient.

Returns
Returns the remainder.

- remquof(x, ±0, quo) returns NaN and stores an unspecified value in the location to which quo points.
- remquof(±∞, y, quo) returns NaN and stores an unspecified value in the location to which quo points.
- remquof(x, y, quo) returns NaN and stores an unspecified value in the location to which quo points if either of x or y is NaN.
- remquof(x, ±∞, quo) returns x and stores zero in the location to which quo points for finite x.

Description
Computes a single-precision floating-point remainder in the same way as the remainderf() function. Argument quo returns part of quotient upon division of x by y. Value quo has the
same sign as $\frac{x}{y}$ and may not be the exact quotient but agrees with the exact quotient in the low order 3 bits.

**Note:**
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

```c
__device__ float rhypotf(float x, float y)
```

Calculate one over the square root of the sum of squares of two arguments.

**Returns**

Returns one over the length of the hypotenuse $\frac{1}{\sqrt{x^2+y^2}}$.

- `rhypotf(x, y)`, `rhypotf(y, x)`, and `rhypotf(x, -y)` are equivalent.
- `rhypotf(±∞, y)` returns +0, even if `y` is a NaN.

**Description**

Calculates one over the length of the hypotenuse of a right triangle whose two sides have lengths `x` and `y` without undue overflow or underflow.

**Note:**
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

```c
__device__ float rintf(float x)
```

Round input to nearest integer value in floating-point.

**Returns**

Returns rounded integer value.

- `rintf(±0)` returns ±0.
- `rintf(±∞)` returns ±∞.

**Description**

Round `x` to the nearest integer value in floating-point format, with halfway cases rounded to the nearest even integer value.
\texttt{__device\_float rnorm3df (float a, float b, float c)}

Calculate one over the square root of the sum of squares of three coordinates.

\textbf{Returns}

Returns one over the length of the 3D vector \( \frac{1}{\sqrt{a^2 + b^2 + c^2}} \).

\begin{itemize}
  \item In the presence of an exactly infinite coordinate \(+0\) is returned, even if there are NaNs.
\end{itemize}

\textbf{Description}

Calculates one over the length of three dimension vector in Euclidean space without undue overflow or underflow.

\begin{quote}
\textbf{Note:}
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
\end{quote}

\texttt{__device\_float rnorm4df (float a, float b, float c, float d)}

Calculate one over the square root of the sum of squares of four coordinates.

\textbf{Returns}

Returns one over the length of the 3D vector \( \frac{1}{\sqrt{a^2 + b^2 + c^2 + d^2}} \).

\begin{itemize}
  \item In the presence of an exactly infinite coordinate \(+0\) is returned, even if there are NaNs.
\end{itemize}

\textbf{Description}

Calculates one over the length of four dimension vector in Euclidean space without undue overflow or underflow.

\begin{quote}
\textbf{Note:}
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
\end{quote}
__device__ float rnormf (int dim, const float *p)
Calculate the reciprocal of square root of the sum of squares of any number of coordinates.

Returns
Returns one over the length of the vector $\frac{1}{\sqrt{p_0^2 + p_1^2 + \ldots + p_{\text{dim}-1}^2}}$.

- In the presence of an exactly infinite coordinate $+\infty$ is returned, even if there are NaNs.

Description
Calculates one over the length of vector $p$, dimension of which is passed as an argument, in Euclidean space without undue overflow or underflow.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float roundf (float x)
Round to nearest integer value in floating-point.

Returns
Returns rounded integer value.

- roundf($\pm 0$) returns $\pm 0$.
- roundf($\pm \infty$) returns $\pm \infty$.

Description
Round $x$ to the nearest integer value in floating-point format, with halfway cases rounded away from zero.

Note:
This function may be slower than alternate rounding methods. See rintf().
__device__ float rsqrtf (float x)
Calculate the reciprocal of the square root of the input argument.

Returns
Returns $1/\sqrt{x}$.
- rsqrtf( $+\infty$ ) returns $+0$.
- rsqrtf( $\pm 0$ ) returns $\pm \infty$.
- rsqrtf($x$) returns NaN if $x$ is less than 0.

Description
Calculate the reciprocal of the nonnegative square root of $x$, $1/\sqrt{x}$.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float scalblnf (float x, long int n)
Scale floating-point input by integer power of two.

Returns
Returns $x \times 2^n$.
- scalblnf( $\pm 0$, $n$ ) returns $\pm 0$.
- scalblnf($x$, 0) returns $x$.
- scalblnf( $\pm \infty$, $n$ ) returns $\pm \infty$.

Description
Scale $x$ by $2^n$ by efficient manipulation of the floating-point exponent.

__device__ float scalbnf (float x, int n)
Scale floating-point input by integer power of two.

Returns
Returns $x \times 2^n$. 
- scalbnf(±0, n) returns ±0.
- scalbnf(x, 0) returns x.
- scalbnf(±∞, n) returns ±∞.

**Description**
Scale x by $2^n$ by efficient manipulation of the floating-point exponent.

**__device__ __RETURN_TYPE signbit (float a)**
Return the sign bit of the input.

**Returns**
Reports the sign bit of all values including infinities, zeros, and NaNs.
- With Visual Studio 2013 host compiler: __RETURN_TYPE is 'bool'. Returns true if and only if a is negative.
- With other host compilers: __RETURN_TYPE is 'int'. Returns a nonzero value if and only if a is negative.

**Description**
Determine whether the floating-point value a is negative.

**__device__ void sincosf (float x, float *sptr, float *cptr)**
Calculate the sine and cosine of the first input argument.

**Returns**
- none

**Description**
Calculate the sine and cosine of the first input argument x (measured in radians). The results for sine and cosine are written into the second argument, sptr, and, respectively, third argument, cptr.

**See also:**
sinf() and cosf().

**Note:**
__device__ void sincospif (float x, float *sptr, float *cptr)
Calculate the sine and cosine of the first input argument \( x \pi \).

Returns

\[ \text{none} \]

Description
Calculate the sine and cosine of the first input argument, \( x \) (measured in radians), \( x \pi \). The results for sine and cosine are written into the second argument, \( \text{sptr} \), and, respectively, third argument, \( \text{cptr} \).

See also:
\[ \text{sinpif()} \] and \[ \text{cospif()} \].

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float sinf (float x)
Calculate the sine of the input argument.

Returns

\[ \text{sinf(-0)} \text{ returns -0}. \]
\[ \text{sinf(0)} \text{ returns NaN}. \]

Description
Calculate the sine of the input argument \( x \) (measured in radians).
__device__ float sinhf (float x)
Calculate the hyperbolic sine of the input argument.

Returns
- \( \sinhf( \pm 0) \) returns \( \pm 0 \).
- \( \sinhf( \pm \infty) \) returns \( \pm \infty \).

Description
Calculate the hyperbolic sine of the input argument \( x \).

---

__device__ float sinpif (float x)
Calculate the sine of the input argument \( x \times \pi \).

Returns
- \( \sinpif( \pm 0) \) returns \( \pm 0 \).
- \( \sinpif( \pm \infty) \) returns NaN.

Description
Calculate the sine of \( x \times \pi \) (measured in radians), where \( x \) is the input argument.

---

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
\section*{Modules

\texttt{CUDA Math API}}

\vspace{0.5cm}

\begin{align*}
\textbf{__device__ float sqrtf (float x)} \\
& \text{Calculate the square root of the input argument.} \\
\end{align*}

\textbf{Returns}

Returns $\sqrt{x}$:

- $\text{sqrtf}(\pm 0)$ returns $\pm 0$.
- $\text{sqrtf}(+\infty)$ returns $+\infty$.
- $\text{sqrtf}(x)$ returns NaN if $x$ is less than 0.

\textbf{Description}

Calculate the nonnegative square root of $x$, $\sqrt{x}$.

\begin{itemize}
\item \textbf{Note:}
\begin{itemize}
\item For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
\end{itemize}
\end{itemize}

\begin{align*}
\textbf{__device__ float tanf (float x)} \\
& \text{Calculate the tangent of the input argument.} \\
\end{align*}

\textbf{Returns}

- $\text{tanf}(\pm 0)$ returns $\pm 0$.
- $\text{tanf}(\pm \infty)$ returns NaN.

\textbf{Description}

Calculate the tangent of the input argument $x$ (measured in radians).

\begin{itemize}
\item \textbf{Note:}
\begin{itemize}
\item For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
\item This function is affected by the \texttt{--use_fast_math} compiler flag. See the CUDA C++ Programming Guide, Mathematical Functions Appendix, Intrinsic Functions section for a complete list of functions affected.
\end{itemize}
\end{itemize}
__device__ float tanhf (float x)
Calculate the hyperbolic tangent of the input argument.

Returns
- \( \tanh(\pm 0) \) returns \( \pm 0 \).
- \( \tanh(\pm \infty) \) returns \( \pm 1 \).

Description
Calculate the hyperbolic tangent of the input argument \( x \).

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float tgammaf (float x)
Calculate the gamma function of the input argument.

Returns
- \( \tgamma(\pm 0) \) returns \( \pm \infty \).
- \( \tgamma(2) \) returns +1.
- \( \tgamma(x) \) returns NaN if \( x < 0 \) and \( x \) is an integer.
- \( \tgamma(\pm \infty) \) returns NaN.
- \( \tgamma(\pm \infty) \) returns +\( \infty \).

Description
Calculate the gamma function of the input argument \( x \), namely the value of \( \int_0^\infty e^{-t}t^{x-1}dt \).

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
__device__ float truncf (float x)
Truncate input argument to the integral part.

Returns
Returns truncated integer value.

- \text{truncf}(±0) \text{ returns } ±0.
- \text{truncf}(±\infty) \text{ returns } ±\infty.

Description
Round \( x \) to the nearest integer value that does not exceed \( x \) in magnitude.

__device__ float y0f (float x)
Calculate the value of the Bessel function of the second kind of order 0 for the input argument.

Returns
Returns the value of the Bessel function of the second kind of order 0.

- \text{y0f}(±0) \text{ returns } -\infty.
- \text{y0f}(x) \text{ returns } \text{NaN} \text{ for } x < 0.
- \text{y0f}(+\infty) \text{ returns } +0.
- \text{y0f}(\text{NaN}) \text{ returns } \text{NaN}.

Description
Calculate the value of the Bessel function of the second kind of order 0 for the input argument \( x \), \( Y_0(x) \).

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float y1f (float x)
Calculate the value of the Bessel function of the second kind of order 1 for the input argument.

Returns
Returns the value of the Bessel function of the second kind of order 1.
Description
Calculate the value of the Bessel function of the second kind of order 1 for the input argument \( x \), \( Y_1(x) \).

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

\[
__
\text{device__}
\text{ float ynf (int n, float x) }
\]
Calculate the value of the Bessel function of the second kind of order \( n \) for the input argument \( x \).

Returns
Returns the value of the Bessel function of the second kind of order \( n \).

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
1.6. Double Precision Mathematical Functions

This section describes double precision mathematical functions. To use these functions you do not need to include any additional header files in your program.

__device__ double acos (double x)

Calculate the arc cosine of the input argument.

Returns
Result will be in radians, in the interval \([0, \pi]\) for \(x\) inside \([-1, +1]\).

- \(\text{acos}(1)\) returns +0.
- \(\text{acos}(x)\) returns NaN for \(x\) outside \([-1, +1]\).

Description
Calculate the principal value of the arc cosine of the input argument \(x\).

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

__device__ double acosh (double x)

Calculate the nonnegative inverse hyperbolic cosine of the input argument.

Returns
Result will be in the interval \([0, +\infty]\).

- \(\text{acosh}(1)\) returns 0.
- \(\text{acosh}(x)\) returns NaN for \(x\) in the interval \([-\infty, 1)\).
- \(\text{acosh}(+\infty)\) returns +\infty.

Description
Calculate the nonnegative inverse hyperbolic cosine of the input argument \(x\).
Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

__device__ double asin (double x)
Calculate the arc sine of the input argument.

Returns
Result will be in radians, in the interval \([- \pi /2, + \pi /2]\) for \(x\) inside \([-1, +1]\).
- \(\text{asin}( \pm 0 )\) returns \(\pm 0\).
- \(\text{asin}(x)\) returns NaN for \(x\) outside \([-1, +1]\).

Description
Calculate the principal value of the arc sine of the input argument \(x\).

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

__device__ double asinh (double x)
Calculate the inverse hyperbolic sine of the input argument.

Returns
- \(\text{asinh}( \pm 0 )\) returns \(\pm 0\).
- \(\text{asinh}( \pm \infty )\) returns \(\pm \infty\).

Description
Calculate the inverse hyperbolic sine of the input argument \(x\).
__device__ double atan (double x)
Calculate the arc tangent of the input argument.

Returns
Result will be in radians, in the interval [- π/2, + π/2].

- atan( ±0 ) returns ±0.
- atan( ±∞ ) returns ±π/2.

Description
Calculate the principal value of the arc tangent of the input argument x.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

__device__ double atan2 (double y, double x)
Calculate the arc tangent of the ratio of first and second input arguments.

Returns
Result will be in radians, in the interval [- π, + π].

- atan2( ±0 , -0) returns ±π.
- atan2( ±0 , +0) returns ±0.
- atan2( ±0 , x) returns ±π for x < 0.
- atan2( ±0 , x) returns ±0 for x > 0.
- atan2(y, ±0 ) returns -π/2 for y < 0.
- atan2(y, ±0 ) returns π/2 for y > 0.
- atan2( ±y , -∞ ) returns ±π for finite y > 0.
- atan2( ±y , +∞ ) returns ±0 for finite y > 0.
- atan2( ±∞ , x) returns ±π/2 for finite x.
- atan2( ±∞ , -∞ ) returns ±3π/4.
- atan2( ±∞ , +∞ ) returns ±π/4.
**Description**

Calculate the principal value of the arc tangent of the ratio of first and second input arguments \( y / x \). The quadrant of the result is determined by the signs of inputs \( y \) and \( x \).

**Note:**

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

**__device__ double atanh (double x)**

Calculate the inverse hyperbolic tangent of the input argument.

**Returns**

- \( \text{atanh} (\pm 0) \) returns \( \pm 0 \).
- \( \text{atanh} (\pm 1) \) returns \( \pm \infty \).
- \( \text{atanh}(x) \) returns NaN for \( x \) outside interval \([-1, 1]\).

**Description**

Calculate the inverse hyperbolic tangent of the input argument \( x \).

**Note:**

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

**__device__ double cbrt (double x)**

Calculate the cube root of the input argument.

**Returns**

Returns \( x^{1/3} \).

- \( \text{cbrt}(\pm 0) \) returns \( \pm 0 \).
- \( \text{cbrt}(\pm \infty) \) returns \( \pm \infty \).

**Description**

Calculate the cube root of \( x \), \( x^{1/3} \).
__device__ double ceil (double x)
Calculate ceiling of the input argument.

Returns
Returns \([x]\) expressed as a floating-point number.

- \(\text{ceil}(\pm0)\) returns \(\pm0\).
- \(\text{ceil}(\pm\infty)\) returns \(\pm\infty\).

Description
Compute the smallest integer value not less than \(x\).

__device__ double copysign (double x, double y)
Create value with given magnitude, copying sign of second value.

Returns
Returns a value with the magnitude of \(x\) and the sign of \(y\).

Description
Create a floating-point value with the magnitude \(x\) and the sign of \(y\).

__device__ double cos (double x)
Calculate the cosine of the input argument.

Returns
- \(\text{cos}(\pm0)\) returns 1.
- \(\text{cos}(\pm\infty)\) returns NaN.

Description
Calculate the cosine of the input argument \(x\) (measured in radians).
__device__ double cosh (double x)

Calculate the hyperbolic cosine of the input argument.

Returns
- cosh(±0) returns 1.
- cosh(±∞) returns +∞.

Description
Calculate the hyperbolic cosine of the input argument x.

__device__ double cospi (double x)

Calculate the cosine of the input argument \( x \times \pi \).

Returns
- cospi(±0) returns 1.
- cospi(±∞) returns NaN.

Description
Calculate the cosine of \( x \times \pi \) (measured in radians), where \( x \) is the input argument.
__device__ double cyl_bessel_i0 (double x)
Calculate the value of the regular modified cylindrical Bessel function of order 0 for the input argument.

Returns
Returns the value of the regular modified cylindrical Bessel function of order 0.

Description
Calculate the value of the regular modified cylindrical Bessel function of order 0 for the input argument \( x \), \( I_0(x) \).

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

__device__ double cyl_bessel_i1 (double x)
Calculate the value of the regular modified cylindrical Bessel function of order 1 for the input argument.

Returns
Returns the value of the regular modified cylindrical Bessel function of order 1.

Description
Calculate the value of the regular modified cylindrical Bessel function of order 1 for the input argument \( x \), \( I_1(x) \).

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

__device__ double erf (double x)
Calculate the error function of the input argument.

Returns
- \( \text{erf}(\pm 0) \) returns \( \pm 0 \).
- \( \text{erf}(\pm \infty) \) returns \( \pm 1 \).
Description

Calculate the value of the error function for the input argument \(x, \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt\).

\[
\text{Note:}
\]
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

\textbf{__device__ double erfc (double x)}

Calculate the complementary error function of the input argument.

\textbf{Returns}

\begin{itemize}
  \item \text{erfc}(-\infty) \text{ returns } 2.
  \item \text{erfc}(+\infty) \text{ returns } +0.
\end{itemize}

\textbf{Description}

Calculate the complementary error function of the input argument \(x, 1 - \text{erf}(x)\).

\textbf{Note:}
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

\textbf{__device__ double erfcinv (double x)}

Calculate the inverse complementary error function of the input argument.

\textbf{Returns}

\begin{itemize}
  \item \text{erfcinv}(\pm 0) \text{ returns } +\infty.
  \item \text{erfcinv}(0) \text{ returns } -\infty.
  \item \text{erfcinv}(x) \text{ returns NaN for } x \text{ outside } [0, 2].
\end{itemize}

\textbf{Description}

Calculate the inverse complementary error function \(\text{erfc}^{-1}(x)\), of the input argument \(x\) in the interval \([0, 2]\).
__device__ double erfcx (double x)
Calculate the scaled complementary error function of the input argument.

Returns

- erfcx( -∞ ) returns +∞.
- erfcx( +∞ ) returns +0.

Description

Calculate the scaled complementary error function of the input argument x, \( e^{x^2} \cdot \text{erfc}(x) \).

__device__ double erfinv (double x)
Calculate the inverse error function of the input argument.

Returns

- erfinv( ±0 ) returns ±0.
- erfinv(1) returns +∞.
- erfinv(-1) returns −∞.
- erfinv(x) returns NaN for x outside [-1, +1].

Description

Calculate the inverse error function \( \text{erf}^{-1}(x) \), of the input argument x in the interval [-1, 1].
__device__ double exp (double x)

Calculate the base $e$ exponential of the input argument.

**Returns**

- $\text{exp}(\pm 0)$ returns 1.
- $\text{exp}(-\infty)$ returns $+0$.
- $\text{exp}(+\infty)$ returns $+\infty$.

**Description**

Calculate $e^x$, the base $e$ exponential of the input argument $x$.

**Note:**

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

__device__ double exp10 (double x)

Calculate the base 10 exponential of the input argument.

**Returns**

- $\text{exp10}(\pm 0)$ returns 1.
- $\text{exp10}(-\infty)$ returns $+0$.
- $\text{exp10}(+\infty)$ returns $+\infty$.

**Description**

Calculate $10^x$, the base 10 exponential of the input argument $x$.

**Note:**

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
`__device__ double exp2 (double x)`
Calculate the base 2 exponential of the input argument.

**Returns**
- exp2(±0) returns 1.
- exp2(−∞) returns 0.
- exp2(+∞) returns +∞.

**Description**
Calculate $2^x$, the base 2 exponential of the input argument $x$.

**Note:**
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

`__device__ double expm1 (double x)`
Calculate the base $e$ exponential of the input argument, minus 1.

**Returns**
- expm1(±0) returns ±0.
- expm1(−∞) returns -1.
- expm1(+∞) returns +∞.

**Description**
Calculate $e^x - 1$, the base $e$ exponential of the input argument $x$, minus 1.

**Note:**
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
__device__ double fabs (double x)
Calculate the absolute value of the input argument.

Returns
Returns the absolute value of the input argument.

- fabs(±∞) returns +∞.
- fabs(±0) returns +0.

Description
Calculate the absolute value of the input argument x.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

__device__ double fdim (double x, double y)
Compute the positive difference between x and y.

Returns
Returns the positive difference between x and y.

- fdim(x, y) returns x - y if x > y.
- fdim(x, y) returns +0 if x ≤ y.

Description
Compute the positive difference between x and y. The positive difference is x - y when x > y and +0 otherwise.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
__device__ double floor (double x)
Calculate the largest integer less than or equal to x.

Returns
Returns \[|x|\] expressed as a floating-point number.

- \(\text{floor}(\pm\infty)\) returns \(\pm\infty\).
- \(\text{floor}(\pm0)\) returns \(\pm0\).

Description
Calculates the largest integer value which is less than or equal to x.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

__device__ double fma (double x, double y, double z)
Compute \(x \times y + z\) as a single operation.

Returns
Returns the rounded value of \(x \times y + z\) as a single operation.

- \(\text{fma}(\pm\infty, \pm0, z)\) returns NaN.
- \(\text{fma}(\pm0, \pm\infty, z)\) returns NaN.
- \(\text{fma}(x, y, -\infty)\) returns NaN if \(x \times y\) is an exact \(+\infty\).
- \(\text{fma}(x, y, +\infty)\) returns NaN if \(x \times y\) is an exact \(-\infty\).

Description
Compute the value of \(x \times y + z\) as a single ternary operation. After computing the value to infinite precision, the value is rounded once.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
__device__ double fmax (double, double)
Determine the maximum numeric value of the arguments.

Returns
Returns the maximum numeric values of the arguments \( x \) and \( y \).

- If both arguments are NaN, returns NaN.
- If one argument is NaN, returns the numeric argument.

Description
Determines the maximum numeric value of the arguments \( x \) and \( y \). Treats NaN arguments as missing data. If one argument is a NaN and the other is legitimate numeric value, the numeric value is chosen.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

__device__ double fmin (double \( x \), double \( y \))
Determine the minimum numeric value of the arguments.

Returns
Returns the minimum numeric value of the arguments \( x \) and \( y \).

- If both arguments are NaN, returns NaN.
- If one argument is NaN, returns the numeric argument.

Description
Determines the minimum numeric value of the arguments \( x \) and \( y \). Treats NaN arguments as missing data. If one argument is a NaN and the other is legitimate numeric value, the numeric value is chosen.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
\_\_device\_\_ double fmod (double x, double y)
Calculate the double-precision floating-point remainder of x / y.

Returns

- Returns the floating-point remainder of x / y.
- \( fmod(\pm0, y) \) returns \( \pm0 \) if y is not zero.
- \( fmod(x, \pm\infty) \) returns x if x is finite.
- \( fmod(x, y) \) returns NaN if x is \( \pm\infty \) or y is zero.
- If either argument is NaN, NaN is returned.

Description

Calculate the double-precision floating-point remainder of x / y. The floating-point remainder of the division operation x / y calculated by this function is exactly the value x - n*y, where n is x / y with its fractional part truncated. The computed value will have the same sign as x, and its magnitude will be less than the magnitude of y.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

\_\_device\_\_ double frexp (double x, int *nptr)
Extract mantissa and exponent of a floating-point value.

Returns

Returns the fractional component m.

- \( frexp(\pm0, nptr) \) returns \( \pm0 \) and stores zero in the location pointed to by nptr.
- \( frexp(\pm\infty, nptr) \) returns \( \pm\infty \) and stores an unspecified value in the location to which nptr points.
- \( frexp(NaN, y) \) returns a NaN and stores an unspecified value in the location to which nptr points.

Description

Decompose the floating-point value x into a component m for the normalized fraction element and another term n for the exponent. The absolute value of m will be greater than or equal to
0.5 and less than 1.0 or it will be equal to 0; \( x = m \cdot 2^n \). The integer exponent \( n \) will be stored in the location to which \( \text{nptr} \) points.

**Note:**
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

```__device__ double hypot (double x, double y)`
Calculate the square root of the sum of squares of two arguments.
```

**Returns**

Returns the length of the hypotenuse \( \sqrt{x^2 + y^2} \).

- hypot(x,y), hypot(y,x), and hypot(x, -y) are equivalent.
- hypot(x, \( \pm 0 \)) is equivalent to fabs(x).
- hypot( \( \pm \infty \), y) returns \( +\infty \), even if y is a NaN.

**Description**

Calculate the length of the hypotenuse of a right triangle whose two sides have lengths \( x \) and \( y \) without undue overflow or underflow.

**Note:**
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

```__device__ int ilogb (double x)`
Compute the unbiased integer exponent of the argument.
```

**Returns**

- If successful, returns the unbiased exponent of the argument.
- \( \text{ilogb}( \pm 0 ) \) returns \( \text{INT_MIN} \).
- \( \text{ilogb}(\text{NaN}) \) returns \( \text{INT_MIN} \).
- \( \text{ilogb}(\pm \infty) \) returns \( \text{INT_MAX} \).
- Note: above behavior does not take into account FP_ILOGB0 nor FP_ILOGBNAN.
Description
Calculates the unbiased integer exponent of the input argument \(x\).

**Note:**
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

__device__ __RETURN_TYPE isfinite (double a)
Determine whether argument is finite.

**Returns**
- With Visual Studio 2013 host compiler: __RETURN_TYPE is 'bool'. Returns true if and only if \(a\) is a finite value.
- With other host compilers: __RETURN_TYPE is 'int'. Returns a nonzero value if and only if \(a\) is a finite value.

Description
Determine whether the floating-point value \(a\) is a finite value (zero, subnormal, or normal and not infinity or NaN).

__device__ __RETURN_TYPE isninf (double a)
Determine whether argument is infinite.

**Returns**
- With Visual Studio 2013 host compiler: Returns true if and only if \(a\) is an infinite value.
- With other host compilers: Returns a nonzero value if and only if \(a\) is an infinite value.

Description
Determine whether the floating-point value \(a\) is an infinite value [positive or negative].
__device__ __RETURN_TYPE isnan (double a)
Determine whether argument is a NaN.

Returns
- With Visual Studio 2013 host compiler: __RETURN_TYPE is ‘bool’. Returns true if and only if a is a NaN value.
- With other host compilers: __RETURN_TYPE is ‘int’. Returns a nonzero value if and only if a is a NaN value.

Description
Determine whether the floating-point value a is a NaN.

__device__ double j0 (double x)
Calculate the value of the Bessel function of the first kind of order 0 for the input argument.

Returns
Returns the value of the Bessel function of the first kind of order 0.
- j0(\pm \infty) returns +0.
- j0(NaN) returns NaN.

Description
Calculate the value of the Bessel function of the first kind of order 0 for the input argument x, \( J_0(x) \).

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

__device__ double j1 (double x)
Calculate the value of the Bessel function of the first kind of order 1 for the input argument.

Returns
Returns the value of the Bessel function of the first kind of order 1.
- j1(\pm 0) returns \pm 0.
- j1(\pm \infty) returns \pm 0.
- \( j_1(\text{NaN}) \) returns NaN.

**Description**

Calculate the value of the Bessel function of the first kind of order 1 for the input argument \( x, J_1(x) \).

**Note:**

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

```c
__device__ double jn (int n, double x)
```

Calculate the value of the Bessel function of the first kind of order \( n \) for the input argument.

**Returns**

Returns the value of the Bessel function of the first kind of order \( n \).

- \( jn(n, \text{NaN}) \) returns NaN.
- \( jn(n, x) \) returns NaN for \( n < 0 \).
- \( jn(n, +\infty) \) returns +0.

**Description**

Calculate the value of the Bessel function of the first kind of order \( n \) for the input argument \( x, J_n(x) \).

**Note:**

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

```c
__device__ double ldexp (double x, int exp)
```

Calculate the value of \( x \cdot 2^{\exp} \).

**Returns**

- \( ldexp(x, \exp) \) is equivalent to \( \text{scalbn}(x, \exp) \).

**Description**

Calculate the value of \( x \cdot 2^{\exp} \) of the input arguments \( x \) and \( \exp \).
__device__ double lgamma (double x)
Calculate the natural logarithm of the absolute value of the gamma function of the input argument.

Returns

- \( \text{lgamma}[1] \) returns +0.
- \( \text{lgamma}[2] \) returns +0.
- \( \text{lgamma}[x] \) returns +\( \infty \) if \( x \leq 0 \) and \( x \) is an integer.
- \( \text{lgamma}[-\infty] \) returns +\( \infty \).
- \( \text{lgamma}[+\infty] \) returns +\( \infty \).

Description

Calculate the natural logarithm of the absolute value of the gamma function of the input argument \( x \), namely the value of \( \log_e \left| \int_0^\infty e^{-t}t^{x-1}dt \right| \).

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

__device__ long long int llrint (double x)
Round input to nearest integer value.

Returns

Returns rounded integer value.

Description

Round \( x \) to the nearest integer value, with halfway cases rounded to the nearest even integer value. If the result is outside the range of the return type, the behavior is undefined.
**__device__** long long int *llround* (double x)

Round to nearest integer value.

**Returns**

Returns rounded integer value.

**Description**

Round x to the nearest integer value, with halfway cases rounded away from zero. If the result is outside the range of the return type, the behavior is undefined.

---

**Note:**

This function may be slower than alternate rounding methods. See [*llrint*](https://docs.nvidia.com/cuda/cuda-c-bridge/index.html).

---

**__device__** double *log* (double x)

Calculate the base e logarithm of the input argument.

**Returns**

- \( \log(\pm 0) \) returns \(-\infty\).
- \( \log(1) \) returns +0.
- \( \log(x) \) returns NaN for \( x < 0 \).
- \( \log(\pm \infty) \) returns \( +\infty \).

**Description**

Calculate the base e logarithm of the input argument \( x \).

---

**Note:**

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

---

**__device__** double *log10* (double x)

Calculate the base 10 logarithm of the input argument.

**Returns**

- \( \log10(\pm 0) \) returns \(-\infty\).
- \( \log_{10}(1) \) returns +0.
- \( \log_{10}(x) \) returns NaN for \( x < 0 \).
- \( \log_{10}( +\infty ) \) returns +\( \infty \).

**Description**

Calculate the base 10 logarithm of the input argument \( x \).

**Note:**
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

```c
__device__ double log1p (double x)
```

Calculate the value of \( \log_e(1 + x) \).

**Returns**

- \( \log_{1p}( \pm 0 ) \) returns \( \pm 0 \).
- \( \log_{1p}(-1) \) returns \( -\infty \).
- \( \log_{1p}(x) \) returns NaN for \( x < -1 \).
- \( \log_{1p}( +\infty ) \) returns +\( \infty \).

**Description**

Calculate the value of \( \log_e(1 + x) \) of the input argument \( x \).

**Note:**
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

```c
__device__ double log2 (double x)
```

Calculate the base 2 logarithm of the input argument.

**Returns**

- \( \log_{2}( \pm 0 ) \) returns \( -\infty \).
- \( \log_{2}(1) \) returns +0.
- \( \log_2(x) \) returns NaN for \( x < 0 \).
- \( \log_2(+\infty) \) returns +\( \infty \).

**Description**

Calculate the base 2 logarithm of the input argument \( x \).

**Note:**

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

__device__ double logb (double x)

Calculate the floating-point representation of the exponent of the input argument.

**Returns**
- \( \log b(\pm 0) \) returns \(-\infty\).
- \( \log b(\pm \infty) \) returns \(+\infty\).

**Description**

Calculate the floating-point representation of the exponent of the input argument \( x \).

**Note:**

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

__device__ long int lrint (double x)

Round input to nearest integer value.

**Returns**

Returns rounded integer value.

**Description**

Round \( x \) to the nearest integer value, with halfway cases rounded to the nearest even integer value. If the result is outside the range of the return type, the behavior is undefined.
__device__ long int lround (double x)
Round to nearest integer value.

Returns
Returns rounded integer value.

Description
Round $x$ to the nearest integer value, with halfway cases rounded away from zero. If the result is outside the range of the return type, the behavior is undefined.

Note:
This function may be slower than alternate rounding methods. See lrint().

__device__ double max (const double a, const float b)
Calculate the maximum value of the input double and float arguments.

Description
Convert float argument $b$ to double, followed by fmax().
Note, this is different from std:: specification

__device__ double max (const float a, const double b)
Calculate the maximum value of the input float and double arguments.

Description
Convert float argument $a$ to double, followed by fmax().
Note, this is different from std:: specification

__device__ double max (const double a, const double b)
Calculate the maximum value of the input double arguments.

Description
Calculate the maximum value of the arguments $a$ and $b$. Behavior is equivalent to fmax() function.
Note, this is different from std:: specification
__device__ double min (const double a, const float b)
Calculate the minimum value of the input double and float arguments.

Description
Convert float argument b to double, followed by fmin().
Note, this is different from std:: specification

__device__ double min (const float a, const double b)
Calculate the minimum value of the input float and double arguments.

Description
Convert float argument a to double, followed by fmin().
Note, this is different from std:: specification

__device__ double min (const double a, const double b)
Calculate the minimum value of the input float arguments.

Description
Calculate the minimum value of the arguments a and b. Behavior is equivalent to fmin() function.
Note, this is different from std:: specification

__device__ double modf (double x, double *iptr)
Break down the input argument into fractional and integral parts.

Returns
- modf(±∞, iptr) returns ±0 and stores ±∞ in the object pointed to by iptr.
- modf(NaN, iptr) stores a NaN in the object pointed to by iptr and returns a NaN.

Description
Break down the argument x into fractional and integral parts. The integral part is stored in the argument iptr. Fractional and integral parts are given the same sign as the argument x.
__device__ double nan (const char *tagp)
Returns “Not a Number” value.

Returns
- nan(tagp) returns NaN.

Description
Return a representation of a quiet NaN. Argument tagp selects one of the possible representations.

__device__ double nearbyint (double x)
Round the input argument to the nearest integer.

Returns
- nearbyint(±0) returns ±0.
- nearbyint(±∞) returns ±∞.

Description
Round argument x to an integer value in double precision floating-point format. Uses round to nearest rounding, with ties rounding to even.
__device__ double nextafter (double x, double y)

Return next representable double-precision floating-point value after argument x in the direction of y.

Returns

- nextafter(x, y) = y if x equals y.
- nextafter(x, y) = NaN if either x or y are NaN.

Description

Calculate the next representable double-precision floating-point value following x in the direction of y. For example, if y is greater than x, nextafter() returns the smallest representable number greater than x.

Note:

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

__device__ double norm (int dim, const double *p)

Calculate the square root of the sum of squares of any number of coordinates.

Returns

Returns the length of the dim-D vector \( \sqrt{p_0^2 + p_1^2 + \ldots + p_{dim-1}^2} \).

- In the presence of an exactly infinite coordinate \( +\infty \) is returned, even if there are NaNs.

Description

Calculate the length of a vector p, dimension of which is passed as an argument without undue overflow or underflow.

Note:

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
__device__ double norm3d (double a, double b, double c)
Calculate the square root of the sum of squares of three coordinates of the argument.

Returns
Returns the length of 3D vector $\sqrt{a^2 + b^2 + c^2}$.

- In the presence of an exactly infinite coordinate $+\infty$ is returned, even if there are NaNs.

Description
Calculate the length of three dimensional vector in Euclidean space without undue overflow or underflow.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

__device__ double norm4d (double a, double b, double c, double d)
Calculate the square root of the sum of squares of four coordinates of the argument.

Returns
Returns the length of 4D vector $\sqrt{a^2 + b^2 + c^2 + d^2}$.

- In the presence of an exactly infinite coordinate $+\infty$ is returned, even if there are NaNs.

Description
Calculate the length of four dimensional vector in Euclidean space without undue overflow or underflow.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
__device__ double normcdf (double x)
Calculate the standard normal cumulative distribution function.

Returns
- \text{normcdf}(+\infty) \text{ returns 1}.
- \text{normcdf}(-\infty) \text{ returns +0}.

Description
Calculate the cumulative distribution function of the standard normal distribution for input argument \(x\), \(\Phi(x)\).

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

__device__ double normcdfinv (double x)
Calculate the inverse of the standard normal cumulative distribution function.

Returns
- \text{normcdfinv}(\pm 0) \text{ returns } -\infty.
- \text{normcdfinv}(1) \text{ returns } +\infty.
- \text{normcdfinv}(x) \text{ returns NaN if } x \text{ is not in the interval } [0,1].

Description
Calculate the inverse of the standard normal cumulative distribution function for input argument \(x\), \(\Phi^{-1}(x)\). The function is defined for input values in the interval \((0,1)\).

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
__device__ double pow (double x, double y)
Calculate the value of first argument to the power of second argument.

Returns

- `pow(±0, y)` returns ±∞ for y an odd integer less than 0.
- `pow(±0, y)` returns +∞ for y less than 0 and not an odd integer.
- `pow(±0, y)` returns ±0 for y an odd integer greater than 0.
- `pow(±0, y)` returns +0 for y > 0 and not an odd integer.
- `pow(-1, ±∞)` returns 1.
- `pow(+1, y)` returns 1 for any y, even a NaN.
- `pow(x, ±0)` returns 1 for any x, even a NaN.
- `pow(x, y)` returns a NaN for finite x < 0 and finite non-integer y.
- `pow(x, -∞)` returns +∞ for |x| < 1.
- `pow(x, -∞)` returns 0 for |x| > 1.
- `pow(x, +∞)` returns 0 for |x| < 1.
- `pow(x, +∞)` returns +∞ for |x| > 1.
- `pow(−∞, y)` returns -0 for y an odd integer less than 0.
- `pow(−∞, y)` returns +0 for y < 0 and not an odd integer.
- `pow(−∞, y)` returns −∞ for y an odd integer greater than 0.
- `pow(−∞, y)` returns +∞ for y > 0 and not an odd integer.
- `pow( +∞ , y)` returns +0 for y < 0.
- `pow( +∞ , y)` returns +∞ for y > 0.

Description
Calculate the value of x to the power of y.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
__device__ double rcbt (double x)
Calculate reciprocal cube root function.

Returns

- rcbt(±0) returns ±∞.
- rcbt(±∞) returns ±0.

Description
Calculate reciprocal cube root function of x.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

__device__ double remainder (double x, double y)
Compute double-precision floating-point remainder.

Returns

- remainder(x, ±0) returns NaN.
- remainder(±∞, y) returns NaN.
- remainder(x, ±∞) returns x for finite x.

Description
Compute double-precision floating-point remainder r of dividing x by y for nonzero y. Thus
\[ r = x - ny \]
The value n is the integer value nearest \( \frac{x}{y} \). In the case when \( |n - \frac{x}{y}| = \frac{1}{2} \), the even n value is chosen.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
__device__ double remquo (double x, double y, int *quo)

Compute double-precision floating-point remainder and part of quotient.

Returns
Returns the remainder.

- remquo(x, ±0, quo) returns NaN and stores an unspecified value in the location to which quo points.
- remquo(±∞, y, quo) returns NaN and stores an unspecified value in the location to which quo points.
- remquo(x, y, quo) returns NaN and stores an unspecified value in the location to which quo points if either of x or y is NaN.
- remquo(x, ±∞, quo) returns x and stores zero in the location to which quo points for finite x.

Description
Compute a double-precision floating-point remainder in the same way as the remainder[] function. Argument quo returns part of quotient upon division of x by y. Value quo has the same sign as \( \frac{x}{y} \) and may not be the exact quotient but agrees with the exact quotient in the low order 3 bits.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

__device__ double rhypot (double x, double y)

Calculate one over the square root of the sum of squares of two arguments.

Returns
Returns one over the length of the hypotenuse \( \frac{1}{\sqrt{x^2 + y^2}} \).

- rhypot(x,y), rhypot(y,x), and rhypot(x, -y) are equivalent.
- rhypot(±∞, y) returns +0, even if y is a NaN.
Description
Calculate one over the length of the hypotenuse of a right triangle whose two sides have lengths \( x \) and \( y \) without undue overflow or underflow.

\[
\text{\__device\__ double rint (double x)}
\]
Round to nearest integer value in floating-point.

Returns
Returns rounded integer value.

- \( \text{rint( } \pm 0 \text{ ) returns } \pm 0. \)
- \( \text{rint( } \pm \infty \text{ ) returns } \pm \infty. \)

Description
Round \( x \) to the nearest integer value in floating-point format, with halfway cases rounded to the nearest even integer value.

\[
\text{\__device\__ double rnorm (int dim, const double *p)}
\]
Calculate the reciprocal of square root of the sum of squares of any number of coordinates.

Returns
Returns one over the length of the vector
\[
\frac{1}{\sqrt{p_0^2 + p_1^2 + \ldots + p_{\text{dim}-1}^2}}.
\]

- In the presence of an exactly infinite coordinate \( +0 \) is returned, even if there are NaNs.

Description
Calculates one over the length of vector \( p \), dimension of which is passed as an argument, in Euclidean space without undue overflow or underflow.

\[
\text{\__device\__ double rint (double x)}
\]
Round to nearest integer value in floating-point.

Returns
Returns rounded integer value.

- \( \text{rint( } \pm 0 \text{ ) returns } \pm 0. \)
- \( \text{rint( } \pm \infty \text{ ) returns } \pm \infty. \)

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
__device__ double rnorm3d (double a, double b, double c)
Calculate one over the square root of the sum of squares of three coordinates.

Returns
Returns one over the length of the 3D vector \( \frac{1}{\sqrt{a^2 + b^2 + c^2}} \).

- In the presence of an exactly infinite coordinate +0 is returned, even if there are NaNs.

Description
Calculate one over the length of three dimensional vector in Euclidean space without undue overflow or underflow.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

__device__ double rnorm4d (double a, double b, double c, double d)
Calculate one over the square root of the sum of squares of four coordinates.

Returns
Returns one over the length of the 3D vector \( \frac{1}{\sqrt{a^2 + b^2 + c^2 + d^2}} \).

- In the presence of an exactly infinite coordinate +0 is returned, even if there are NaNs.

Description
Calculate one over the length of four dimensional vector in Euclidean space without undue overflow or underflow.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
__device__ double round (double x)
Round to nearest integer value in floating-point.

Returns
Returns rounded integer value.
- `round( ±0 )` returns ±0.
- `round( ±∞ )` returns ±∞.

Description
Round x to the nearest integer value in floating-point format, with halfway cases rounded away from zero.

Note:
This function may be slower than alternate rounding methods. See `rintf`.

__device__ double rsqrt (double x)
Calculate the reciprocal of the square root of the input argument.

Returns
Returns \( 1/\sqrt{x} \).
- `rsqrt( +∞ )` returns +0.
- `rsqrt( ±0 )` returns ±∞.
- `rsqrt(x)` returns NaN if x is less than 0.

Description
Calculate the reciprocal of the nonnegative square root of x, \( 1/\sqrt{x} \).

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
__device__ double scalbln (double x, long int n)
Scale floating-point input by integer power of two.

Returns
Returns \( x \times 2^n \).

- \( \text{scalbln}(\pm 0, n) \) returns \( \pm 0 \).
- \( \text{scalbln}(x, 0) \) returns \( x \).
- \( \text{scalbln}(\pm \infty, n) \) returns \( \pm \infty \).

Description
Scale \( x \) by \( 2^n \) by efficient manipulation of the floating-point exponent.

__device__ double scalbn (double x, int n)
Scale floating-point input by integer power of two.

Returns
Returns \( x \times 2^n \).

- \( \text{scalbn}(\pm 0, n) \) returns \( \pm 0 \).
- \( \text{scalbn}(x, 0) \) returns \( x \).
- \( \text{scalbn}(\pm \infty, n) \) returns \( \pm \infty \).

Description
Scale \( x \) by \( 2^n \) by efficient manipulation of the floating-point exponent.

__device__ ___RETURN_TYPE signbit (double a)
Return the sign bit of the input.

Returns
Reports the sign bit of all values including infinities, zeros, and NaNs.

- With Visual Studio 2013 host compiler: ___RETURN_TYPE is ‘bool’. Returns true if and only if \( a \) is negative.
- With other host compilers: ___RETURN_TYPE is ‘int’. Returns a nonzero value if and only if \( a \) is negative.
Description
Determine whether the floating-point value \( a \) is negative.

\[
\text{__device__ double sin (double x)}
\]
Calculate the sine of the input argument.

Returns
\[
\begin{align*}
\sin(\pm 0) & \text{ returns } \pm 0. \\
\sin(\pm \infty) & \text{ returns NaN.}
\end{align*}
\]

Description
Calculate the sine of the input argument \( x \) (measured in radians).

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

\[
\text{__device__ void sincos (double x, double *sptr, double *cptr)}
\]
Calculate the sine and cosine of the first input argument.

Returns
\[
\text{none}
\]

Description
Calculate the sine and cosine of the first input argument \( x \) (measured in radians). The results for sine and cosine are written into the second argument, \( \text{sptr} \), and, respectively, third argument, \( \text{cptr} \).

See also:
sin() and cos().

Note:
__device__ void sincospi (double x, double *sptr, double *cptr)
Calculate the sine and cosine of the first input argument \(x \pi\).

Returns

- none

Description

Calculate the sine and cosine of the first input argument, \(x\) (measured in radians), \(x \pi\). The results for sine and cosine are written into the second argument, \(\text{sptr}\), and, respectively, third argument, \(\text{cptr}\).

See also:

sinpi() and cospi().

Note:

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

__device__ double sinh (double x)
Calculate the hyperbolic sine of the input argument.

Returns

- \(\text{sinh}(\pm 0)\) returns \(\pm 0\).
- \(\text{sinh}(\pm \infty)\) returns \(\pm \infty\).

Description

Calculate the hyperbolic sine of the input argument \(x\).

Note:

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
**__device__ double sinpi (double x)**
Calculate the sine of the input argument $x \times \pi$.

**Returns**
- $\sin\pi(\pm 0)$ returns $\pm 0$.
- $\sin\pi(\pm \infty)$ returns NaN.

**Description**
Calculate the sine of $x \times \pi$ [measured in radians], where $x$ is the input argument.

**Note:**
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

**__device__ double sqrt (double x)**
Calculate the square root of the input argument.

**Returns**
Returns $\sqrt{x}$.
- $\sqrt{\pm 0}$ returns $\pm 0$.
- $\sqrt{+\infty}$ returns $+\infty$.
- $\sqrt{x}$ returns NaN if $x$ is less than 0.

**Description**
Calculate the nonnegative square root of $x$, $\sqrt{x}$.

**Note:**
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
__device__ double tan (double x)
Calculate the tangent of the input argument.

Returns

- \( \tan( \pm 0 ) \) returns \( \pm 0 \).
- \( \tan( \pm \infty ) \) returns NaN.

Description
Calculate the tangent of the input argument \( x \) (measured in radians).

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

__device__ double tanh (double x)
Calculate the hyperbolic tangent of the input argument.

Returns

- \( \tanh( \pm 0 ) \) returns \( \pm 0 \).
- \( \tanh( \pm \infty ) \) returns \( \pm 1 \).

Description
Calculate the hyperbolic tangent of the input argument \( x \).

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

__device__ double tgamma (double x)
Calculate the gamma function of the input argument.

Returns

- \( \text{tgamma}( \pm 0 ) \) returns \( \pm \infty \).
- \( \text{tgamma}(2) \) returns +1.
• \texttt{tgamma(x)} returns NaN if \( x < 0 \) and \( x \) is an integer.
• \texttt{tgamma(-\infty)} returns NaN.
• \texttt{tgamma(+\infty)} returns +\infty.

\textbf{Description}

Calculate the gamma function of the input argument \( x \), namely the value of \( \int_{0}^{\infty} e^{-t}t^{x-1}dt \).

\begin{quote}
\textbf{Note:}
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
\end{quote}

\textbf{\texttt{__device__ double trunc (double x)}}

Truncate input argument to the integral part.

\textbf{Returns}

Returns truncated integer value.

• \texttt{trunc(\pm0)} returns \pm0.
• \texttt{trunc(\pm\infty)} returns \pm\infty.

\textbf{Description}

Round \( x \) to the nearest integer value that does not exceed \( x \) in magnitude.

\textbf{\texttt{__device__ double y0 (double x)}}

Calculate the value of the Bessel function of the second kind of order 0 for the input argument.

\textbf{Returns}

Returns the value of the Bessel function of the second kind of order 0.

• \texttt{y0(\pm0)} returns \(-\infty\).
• \texttt{y0(x)} returns NaN for \( x < 0 \).
• \texttt{y0(+\infty)} returns +0.
• \texttt{y0(NaN)} returns NaN.
Description
Calculate the value of the Bessel function of the second kind of order 0 for the input argument \( x \), \( Y_0(x) \).

__device__ double y1 (double x)
Calculate the value of the Bessel function of the second kind of order 1 for the input argument.

Returns
Returns the value of the Bessel function of the second kind of order 1.

- \( y1(\pm 0) \) returns \(-\infty\).
- \( y1(x) \) returns NaN for \( x < 0 \).
- \( y1(\infty) \) returns +0.
- \( y1(\text{NaN}) \) returns NaN.

Description
Calculate the value of the Bessel function of the second kind of order 1 for the input argument \( x \), \( Y_1(x) \).

__device__ double yn (int n, double x)
Calculate the value of the Bessel function of the second kind of order \( n \) for the input argument.

Returns
Returns the value of the Bessel function of the second kind of order \( n \).

- \( yn(n, \infty) \) returns NaN for \( n < 0 \).
- \( yn(n, \pm 0) \) returns \(-\infty\).
yn(n, x) returns NaN for x < 0.
yn(n, +∞) returns +0.
yn(n, NaN) returns NaN.

Description
Calculate the value of the Bessel function of the second kind of order n for the input argument x, \( Y_n(x) \).

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

1.7. Integer Mathematical Functions

This section describes integer mathematical functions. To use these functions you do not need to include any additional header files in your program.

__device__ int abs (int a)
Calculate the absolute value of the input int argument.

Description
Calculate the absolute value of the input argument a.

__device__ long int labs (long int a)
Calculate the absolute value of the input long int argument.

Description
Calculate the absolute value of the input argument a.

__device__ long long int llabs (long long int a)
Calculate the absolute value of the input long long int argument.

Description
Calculate the absolute value of the input argument a.
__device__ long long int llmax (const long long int a, const long long int b)
Calculate the maximum value of the input long long int arguments.

Description
Calculate the maximum value of the arguments a and b.

__device__ long long int llmin (const long long int a, const long long int b)
Calculate the minimum value of the input long long int arguments.

Description
Calculate the minimum value of the arguments a and b.

__device__ unsigned long long int max (const unsigned long long int a, const long long int b)
Calculate the maximum value of the input unsigned long long int and long long int arguments.

Description
Calculate the maximum value of the arguments a and b, perform integer promotion first.

__device__ unsigned long long int max (const long long int a, const unsigned long long int b)
Calculate the maximum value of the input long long int and unsigned long long int arguments.

Description
Calculate the maximum value of the arguments a and b, perform integer promotion first.
\_\_device\_\_ unsigned long long int max (const unsigned long long int a, const unsigned long long int b)
Calculate the maximum value of the input unsigned long long int arguments.

Description
Calculate the maximum value of the arguments a and b.

\_\_device\_\_ long long int max (const long long int a, const long long int b)
Calculate the maximum value of the input long long int arguments.

Description
Calculate the maximum value of the arguments a and b.

\_\_device\_\_ unsigned long int max (const unsigned long int a, const long int b)
Calculate the maximum value of the input unsigned long int and long int arguments.

Description
Calculate the maximum value of the arguments a and b, perform integer promotion first.

\_\_device\_\_ unsigned long int max (const long int a, const unsigned long int b)
Calculate the maximum value of the input long int and unsigned long int arguments.

Description
Calculate the maximum value of the arguments a and b, perform integer promotion first.

\_\_device\_\_ unsigned long int max (const unsigned long int a, const unsigned long int b)
Calculate the maximum value of the input unsigned long int arguments.

Description
Calculate the maximum value of the arguments a and b.
`__device__ long int max (const long int a, const long int b)`
Calculate the maximum value of the input long int arguments.

**Description**
Calculate the maximum value of the arguments `a` and `b`.

`__device__ unsigned int max (const unsigned int a, const int b)`
Calculate the maximum value of the input unsigned int and int arguments.

**Description**
Calculate the maximum value of the arguments `a` and `b`, perform integer promotion first.

`__device__ unsigned int max (const int a, const unsigned int b)`
Calculate the maximum value of the input int and unsigned int arguments.

**Description**
Calculate the maximum value of the arguments `a` and `b`, perform integer promotion first.

`__device__ unsigned int max (const unsigned int a, const unsigned int b)`
Calculate the maximum value of the input unsigned int arguments.

**Description**
Calculate the maximum value of the arguments `a` and `b`.

`__device__ int max (const int a, const int b)`
Calculate the maximum value of the input int arguments.

**Description**
Calculate the maximum value of the arguments `a` and `b`. 
__device__ unsigned long long int min (const unsigned long long int a, const long long int b)
Calculate the minimum value of the input unsigned long long int and long long int arguments.

Description
Calculate the minimum value of the arguments a and b, perform integer promotion first.

__device__ unsigned long long int min (const long long int a, const unsigned long long int b)
Calculate the minimum value of the input long long int and unsigned long long int arguments.

Description
Calculate the minimum value of the arguments a and b, perform integer promotion first.

__device__ unsigned long long int min (const unsigned long long int a, const unsigned long long int b)
Calculate the minimum value of the input unsigned long long int arguments.

Description
Calculate the minimum value of the arguments a and b.

__device__ long long int min (const long long int a, const long long int b)
Calculate the minimum value of the input long long int arguments.

Description
Calculate the minimum value of the arguments a and b.
__device__ unsigned long int min (const unsigned long int a, const long int b)
Calculate the minimum value of the input unsigned long int and long int arguments.

Description
Calculate the minimum value of the arguments a and b, perform integer promotion first.

__device__ unsigned long int min (const long int a, const unsigned long int b)
Calculate the minimum value of the input long int and unsigned long int arguments.

Description
Calculate the minimum value of the arguments a and b, perform integer promotion first.

__device__ unsigned long int min (const unsigned long int a, const unsigned long int b)
Calculate the minimum value of the input unsigned long int arguments.

Description
Calculate the minimum value of the arguments a and b.

__device__ long int min (const long int a, const long int b)
Calculate the minimum value of the input long int arguments.

Description
Calculate the minimum value of the arguments a and b.

__device__ unsigned int min (const unsigned int a, const int b)
Calculate the minimum value of the input unsigned int and int arguments.

Description
Calculate the minimum value of the arguments a and b, perform integer promotion first.
__device__ unsigned int min (const int a, const unsigned int b)
Calculate the minimum value of the input int and unsigned int arguments.

Description
Calculate the minimum value of the arguments a and b, perform integer promotion first.

__device__ unsigned int min (const unsigned int a, const unsigned int b)
Calculate the minimum value of the input unsigned int arguments.

Description
Calculate the minimum value of the arguments a and b.

__device__ int min (const int a, const int b)
Calculate the minimum value of the input int arguments.

Description
Calculate the minimum value of the arguments a and b.

__device__ unsigned long long int ullmax (const unsigned long long int a, const unsigned long long int b)
Calculate the maximum value of the input unsigned long long int arguments.

Description
Calculate the maximum value of the arguments a and b.

__device__ unsigned long long int ullmin (const unsigned long long int a, const unsigned long long int b)
Calculate the minimum value of the input unsigned long long int arguments.

Description
Calculate the minimum value of the arguments a and b.
__device__ unsigned int umax (const unsigned int a, const unsigned int b)
Calculate the maximum value of the input unsigned int arguments.

Description
Calculate the maximum value of the arguments a and b.

__device__ unsigned int umin (const unsigned int a, const unsigned int b)
Calculate the minimum value of the input unsigned int arguments.

Description
Calculate the minimum value of the arguments a and b.

1.8. Single Precision Intrinsics
This section describes single precision intrinsic functions that are only supported in device code. To use these functions you do not need to include any additional header files in your program.

__device__ float __cosf (float x)
Calculate the fast approximate cosine of the input argument.

Returns
Returns the approximate cosine of x.

Description
Calculate the fast approximate cosine of the input argument x, measured in radians.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Intrinsic Functions section.
__device__ float __exp10f (float x)
Calculate the fast approximate base 10 exponential of the input argument.

**Returns**
Returns an approximation to $10^x$.

**Description**
Calculate the fast approximate base 10 exponential of the input argument $x$, $10^x$.

**Note:**
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Intrinsic Functions section.

__device__ float __expf (float x)
Calculate the fast approximate base $e$ exponential of the input argument.

**Returns**
Returns an approximation to $e^x$.

**Description**
Calculate the fast approximate base $e$ exponential of the input argument $x$, $e^x$.

**Note:**
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Intrinsic Functions section.

__device__ float __fadd_rd (float x, float y)
Add two floating-point values in round-down mode.

**Returns**
Returns $x + y$.

**Description**
Compute the sum of $x$ and $y$ in round-down (to negative infinity) mode.
Note:

- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.

__device__ float __fadd_rn (float x, float y)

Add two floating-point values in round-to-nearest-even mode.

Returns

Returns x + y.

Description

Compute the sum of x and y in round-to-nearest-even rounding mode.

Note:

- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.

__device__ float __fadd_ru (float x, float y)

Add two floating-point values in round-up mode.

Returns

Returns x + y.

Description

Compute the sum of x and y in round-up (to positive infinity) mode.

Note:

- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.
__device__ float __fadd_rz (float x, float y)
Add two floating-point values in round-towards-zero mode.

Returns
Returns x + y.

Description
Compute the sum of x and y in round-towards-zero mode.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.

__device__ float __fdiv_rd (float x, float y)
Divide two floating-point values in round-down mode.

Returns
Returns x / y.

Description
Divide two floating-point values x by y in round-down (to negative infinity) mode.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float __fdiv_rn (float x, float y)
Divide two floating-point values in round-to-nearest-even mode.

Returns
Returns x / y.

Description
Divide two floating-point values x by y in round-to-nearest-even mode.
__device__ float __fdiv_ru (float x, float y)
Divide two floating-point values in round-up mode.

Returns
Returns x / y.

Description
Divide two floating-point values x by y in round-up (to positive infinity) mode.

__device__ float __fdiv_rz (float x, float y)
Divide two floating-point values in round-towards-zero mode.

Returns
Returns x / y.

Description
Divide two floating-point values x by y in round-towards-zero mode.

__device__ float __fdividef (float x, float y)
Calculate the fast approximate division of the input arguments.

Returns
Returns x / y.
__fdividef(∞, y) returns NaN for \(2^{126} < |y| < 2^{128}\).

__fdividef(x, y) returns 0 for \(2^{126} < |y| < 2^{128}\) and finite \(x\).

Description
Calculate the fast approximate division of \(x\) by \(y\).

**Note:**
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Intrinsic Functions section.

```c
__device__ float __fmaf_ieee_rd (float x, float y, float z)
```

Compute fused multiply-add operation in round-down mode, ignore -ftz=true compiler flag.

Description
Behavior is the same as __fmaf_rd(x, y, z), the difference is in handling denormalized inputs and outputs: -ftz compiler flag has no effect.

```c
__device__ float __fmaf_ieee_rn (float x, float y, float z)
```

Compute fused multiply-add operation in round-to-nearest-even mode, ignore -ftz=true compiler flag.

Description
Behavior is the same as __fmaf_rn(x, y, z), the difference is in handling denormalized inputs and outputs: -ftz compiler flag has no effect.

```c
__device__ float __fmaf_ieee_ru (float x, float y, float z)
```

Compute fused multiply-add operation in round-up mode, ignore -ftz=true compiler flag.

Description
Behavior is the same as __fmaf_ru(x, y, z), the difference is in handling denormalized inputs and outputs: -ftz compiler flag has no effect.
__device__ float __fmaf_ieee_rz (float x, float y, float z)
Compute fused multiply-add operation in round-towards-zero mode, ignore `-ftz=true` compiler flag.

**Description**
Behavior is the same as __fmaf_rz(x, y, z), the difference is in handling denormalized inputs and outputs: `-ftz` compiler flag has no effect.

__device__ float __fmaf_rd (float x, float y, float z)
Compute \(x \times y + z\) as a single operation, in round-down mode.

**Returns**
Returns the rounded value of \(x \times y + z\) as a single operation.

- \(\text{fmaf}(\pm \infty, \pm 0, z)\) returns NaN.
- \(\text{fmaf}(\pm 0, \pm \infty, z)\) returns NaN.
- \(\text{fmaf}(x, y, -\infty)\) returns NaN if \(x \times y\) is an exact \(+\infty\).
- \(\text{fmaf}(x, y, +\infty)\) returns NaN if \(x \times y\) is an exact \(-\infty\).

**Description**
Computes the value of \(x \times y + z\) as a single ternary operation, rounding the result once in round-down (to negative infinity) mode.

**Note:**
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float __fmaf_rn (float x, float y, float z)
Compute \(x \times y + z\) as a single operation, in round-to-nearest-even mode.

**Returns**
Returns the rounded value of \(x \times y + z\) as a single operation.

- \(\text{fmaf}(\pm \infty, \pm 0, z)\) returns NaN.
- \(\text{fmaf}(\pm 0, \pm \infty, z)\) returns NaN.
• fmaf(x, y, −∞) returns NaN if x × y is an exact +∞.
• fmaf(x, y, +∞) returns NaN if x × y is an exact −∞.

Description
Computes the value of x × y + z as a single ternary operation, rounding the result once in round-to-nearest-even mode.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float __fmaf_ru (float x, float y, float z)
Compute x × y + z as a single operation, in round-up mode.

Returns
Returns the rounded value of x × y + z as a single operation.
• fmaf(±∞, ±0, z) returns NaN.
• fmaf(±0, ±∞, z) returns NaN.
• fmaf(x, y, −∞) returns NaN if x × y is an exact +∞.
• fmaf(x, y, +∞) returns NaN if x × y is an exact −∞.

Description
Computes the value of x × y + z as a single ternary operation, rounding the result once in round-up (to positive infinity) mode.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float __fmaf_rz (float x, float y, float z)
Compute x × y + z as a single operation, in round-towards-zero mode.

Returns
Returns the rounded value of x × y + z as a single operation.
• fmaf(±∞, ±0, z) returns NaN.
- \( \text{fmaf}( \pm 0, \pm \infty, z) \) returns NaN.
- \( \text{fmaf}(x, y, -\infty) \) returns NaN if \( xy \) is an exact \( +\infty \).
- \( \text{fmaf}(x, y, +\infty) \) returns NaN if \( xy \) is an exact \( -\infty \).

**Description**

Computes the value of \( xy + z \) as a single ternary operation, rounding the result once in round-towards-zero mode.

**Note:**

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

```c
__device__ float __fmul_rd (float x, float y)
```

Multiply two floating-point values in round-down mode.

**Returns**

Returns \( x \times y \).

**Description**

Compute the product of \( x \) and \( y \) in round-down (to negative infinity) mode.

**Note:**

- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.

```c
__device__ float __fmul_rn (float x, float y)
```

Multiply two floating-point values in round-to-nearest-even mode.

**Returns**

Returns \( x \times y \).

**Description**

Compute the product of \( x \) and \( y \) in round-to-nearest-even mode.
Note:

- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.

**__device__ float __fmul_ru (float x, float y)**
Multiply two floating-point values in round-up mode.

**Returns**
Returns \( x \times y \).

**Description**
Compute the product of \( x \) and \( y \) in round-up (to positive infinity) mode.

Note:

- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.

**__device__ float __fmul_rz (float x, float y)**
Multiply two floating-point values in round-towards-zero mode.

**Returns**
Returns \( x \times y \).

**Description**
Compute the product of \( x \) and \( y \) in round-towards-zero mode.

Note:

- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.
__device__ float __frcp_rd (float x)
Compute \( \frac{1}{x} \) in round-down mode.

Returns

Returns \( \frac{1}{x} \).

Description

Compute the reciprocal of \( x \) in round-down [to negative infinity] mode.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float __frcp_rn (float x)
Compute \( \frac{1}{x} \) in round-to-nearest-even mode.

Returns

Returns \( \frac{1}{x} \).

Description

Compute the reciprocal of \( x \) in round-to-nearest-even mode.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float __frcp_ru (float x)
Compute \( \frac{1}{x} \) in round-up mode.

Returns

Returns \( \frac{1}{x} \).
Description
Compute the reciprocal of x in round-up (to positive infinity) mode.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float __frcp_rz (float x)
Compute $\frac{1}{x}$ in round-towards-zero mode.

Returns
Returns $\frac{1}{x}$.

Description
Compute the reciprocal of x in round-towards-zero mode.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float __frsqrt_rn (float x)
Compute $\frac{1}{\sqrt{x}}$ in round-to-nearest-even mode.

Returns
Returns $\frac{1}{\sqrt{x}}$.

Description
Compute the reciprocal square root of x in round-to-nearest-even mode.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
__device__ float __fsqrt_rd (float x)
Compute $\sqrt{x}$ in round-down mode.

Returns
Returns $\sqrt{x}$.

Description
Compute the square root of $x$ in round-down (to negative infinity) mode.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float __fsqrt_rn (float x)
Compute $\sqrt{x}$ in round-to-nearest-even mode.

Returns
Returns $\sqrt{x}$.

Description
Compute the square root of $x$ in round-to-nearest-even mode.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float __fsqrt_ru (float x)
Compute $\sqrt{x}$ in round-up mode.

Returns
Returns $\sqrt{x}$. 
Description
Compute the square root of $x$ in round-up (to positive infinity) mode.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float __fsqrt_rz (float x)
Compute $\sqrt{x}$ in round-towards-zero mode.

Returns
Returns $\sqrt{x}$.

Description
Compute the square root of $x$ in round-towards-zero mode.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float __fsub_rd (float x, float y)
Subtract two floating-point values in round-down mode.

Returns
Returns $x - y$.

Description
Compute the difference of $x$ and $y$ in round-down (to negative infinity) mode.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.
__device__ float __fsub_rn (float x, float y)
Subtract two floating-point values in round-to-nearest-even mode.

**Returns**
Returns \(x - y\).

**Description**
Compute the difference of \(x\) and \(y\) in round-to-nearest-even rounding mode.

**Note:**
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.

__device__ float __fsub_ru (float x, float y)
Subtract two floating-point values in round-up mode.

**Returns**
Returns \(x - y\).

**Description**
Compute the difference of \(x\) and \(y\) in round-up (to positive infinity) mode.

**Note:**
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.

__device__ float __fsub_rz (float x, float y)
Subtract two floating-point values in round-towards-zero mode.

**Returns**
Returns \(x - y\).
**Description**
Compute the difference of \( x \) and \( y \) in round-towards-zero mode.

**Note:**
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.

```c
__device__ float __log10f (float x)
```
Calculate the fast approximate base 10 logarithm of the input argument.

**Returns**
Returns an approximation to \( \log_{10}(x) \).

**Description**
Calculate the fast approximate base 10 logarithm of the input argument \( x \).

**Note:**
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Intrinsic Functions section.

```c
__device__ float __log2f (float x)
```
Calculate the fast approximate base 2 logarithm of the input argument.

**Returns**
Returns an approximation to \( \log_2(x) \).

**Description**
Calculate the fast approximate base 2 logarithm of the input argument \( x \).

**Note:**
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Intrinsic Functions section.
__device__ float __logf (float x)
Calculate the fast approximate base \(e\) logarithm of the input argument.

Returns
Returns an approximation to \(\log_e(x)\).

Description
Calculate the fast approximate base \(e\) logarithm of the input argument \(x\).

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Intrinsic Functions section.

__device__ float __powf (float x, float y)
Calculate the fast approximate of \(x^y\).

Returns
Returns an approximation to \(x^y\).

Description
Calculate the fast approximate of \(x\), the first input argument, raised to the power of \(y\), the second input argument, \(x^y\).

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Intrinsic Functions section.

__device__ float __saturatef (float x)
Clamp the input argument to \([+0.0, 1.0]\).

Returns
- __saturatef(x) returns 0 if \(x < 0\).
- __saturatef(x) returns 1 if \(x > 1\).
- __saturatef(x) returns \(x\) if \(0 \leq x \leq 1\).
- __saturatef(NaN) returns 0.

**Description**
Clamp the input argument \( x \) to be within the interval \([+0.0, 1.0]\).

```c
__device__ void __sincosf (float x, float *sptr, float *cptr)
```
Calculate the fast approximate of sine and cosine of the first input argument.

**Returns**
- none

**Description**
Calculate the fast approximate of sine and cosine of the first input argument \( x \) (measured in radians). The results for sine and cosine are written into the second argument, \( sptr \), and, respectively, third argument, \( cptr \).

**Note:**
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Intrinsic Functions section.
- Denorm input/output is flushed to sign preserving 0.0.

```c
__device__ float __sinf (float x)
```
Calculate the fast approximate sine of the input argument.

**Returns**
Returns the approximate sine of \( x \).

**Description**
Calculate the fast approximate sine of the input argument \( x \), measured in radians.

**Note:**
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Intrinsic Functions section.
- Output in the denormal range is flushed to sign preserving 0.0.
__device__ float __tanf (float x)
Calculate the fast approximate tangent of the input argument.

Returns
Returns the approximate tangent of x.

Description
Calculate the fast approximate tangent of the input argument x, measured in radians.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Intrinsic Functions section.
- The result is computed as the fast divide of __sinf() by __cosf(). Denormal output is flushed to sign-preserving 0.0.

1.9. Double Precision Intrinsics
This section describes double precision intrinsic functions that are only supported in device code. To use these functions you do not need to include any additional header files in your program.

__device__ double __dadd_rd (double x, double y)
Add two floating-point values in round-down mode.

Returns
Returns x + y.

Description
Adds two floating-point values x and y in round-down (to negative infinity) mode.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.
__device__ double __dadd_rn (double x, double y)

Add two floating-point values in round-to-nearest-even mode.

Returns
Returns \( x + y \).

Description
Adds two floating-point values \( x \) and \( y \) in round-to-nearest-even mode.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.

__device__ double __dadd_ru (double x, double y)

Add two floating-point values in round-up mode.

Returns
Returns \( x + y \).

Description
Adds two floating-point values \( x \) and \( y \) in round-up (to positive infinity) mode.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.

__device__ double __dadd_rz (double x, double y)

Add two floating-point values in round-towards-zero mode.

Returns
Returns \( x + y \).
Description
Adds two floating-point values $x$ and $y$ in round-towards-zero mode.

**Note:**
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.

__device__ double __ddiv_rd (double x, double y)
Divide two floating-point values in round-down mode.

**Returns**
Returns $x / y$.

**Description**
Divides two floating-point values $x$ by $y$ in round-down to negative infinity mode.

**Note:**
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- Requires compute capability $\geq 2.0$.

__device__ double __ddiv_rn (double x, double y)
Divide two floating-point values in round-to-nearest-even mode.

**Returns**
Returns $x / y$.

**Description**
Divides two floating-point values $x$ by $y$ in round-to-nearest-even mode.

**Note:**
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
__device__ double __ddiv_ru (double x, double y)
Divide two floating-point values in round-up mode.

Returns
Returns x / y.

Description
Divides two floating-point values x by y in round-up (to positive infinity) mode.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- Requires compute capability >= 2.0.

__device__ double __ddiv_rz (double x, double y)
Divide two floating-point values in round-towards-zero mode.

Returns
Returns x / y.

Description
Divides two floating-point values x by y in round-towards-zero mode.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- Requires compute capability >= 2.0.

__device__ double __dmul_rd (double x, double y)
Multiply two floating-point values in round-down mode.

Returns
Returns x * y.
Description

Multiplies two floating-point values \( x \) and \( y \) in round-down (to negative infinity) mode.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.

\[
\text{__device__ double __dmul_rn (double x, double y)}
\]
Multiply two floating-point values in round-to-nearest-even mode.

Returns

Returns \( x \times y \).

Description

Multiplies two floating-point values \( x \) and \( y \) in round-to-nearest-even mode.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.

\[
\text{__device__ double __dmul_ru (double x, double y)}
\]
Multiply two floating-point values in round-up mode.

Returns

Returns \( x \times y \).

Description

Multiplies two floating-point values \( x \) and \( y \) in round-up (to positive infinity) mode.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
__device__ double __dmul_rz (double x, double y)
Multiply two floating-point values in round-towards-zero mode.

Returns
Returns x * y.

Description
Multiplies two floating-point values x and y in round-towards-zero mode.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.

__device__ double __drcp_rd (double x)
Compute $\frac{1}{x}$ in round-down mode.

Returns
Returns $\frac{1}{x}$.

Description
Compute the reciprocal of x in round-down (to negative infinity) mode.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- Requires compute capability >= 2.0.
__device__ double __drcp_rn (double x)
Compute $\frac{1}{x}$ in round-to-nearest-even mode.

Returns
Returns $\frac{1}{x}$.

Description
Compute the reciprocal of $x$ in round-to-nearest-even mode.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- Requires compute capability $\geq 2.0$.

__device__ double __drcp_ru (double x)
Compute $\frac{1}{x}$ in round-up mode.

Returns
Returns $\frac{1}{x}$.

Description
Compute the reciprocal of $x$ in round-up (to positive infinity) mode.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- Requires compute capability $\geq 2.0$.

__device__ double __drcp_rz (double x)
Compute $\frac{1}{x}$ in round-towards-zero mode.

Returns
Returns $\frac{1}{x}$.
**Description**
Compute the reciprocal of \( x \) in round-towards-zero mode.

**Note:**
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- Requires compute capability >= 2.0.

```c
__device__ double __dsqrt_rd (double x)
```
Compute \( \sqrt{x} \) in round-down mode.

**Returns**
Returns \( \sqrt{x} \).

**Description**
Compute the square root of \( x \) in round-down (to negative infinity) mode.

**Note:**
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- Requires compute capability >= 2.0.

```c
__device__ double __dsqrt_rn (double x)
```
Compute \( \sqrt{x} \) in round-to-nearest-even mode.

**Returns**
Returns \( \sqrt{x} \).

**Description**
Compute the square root of \( x \) in round-to-nearest-even mode.

**Note:**
__device__ double __dsqrt_ru (double x)

Compute $\sqrt{x}$ in round-up mode.

Returns

Returns $\sqrt{x}$.

Description

Compute the square root of $x$ in round-up (to positive infinity) mode.

Note:

- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- Requires compute capability $\geq$ 2.0.

__device__ double __dsqrt_rz (double x)

Compute $\sqrt{x}$ in round-towards-zero mode.

Returns

Returns $\sqrt{x}$.

Description

Compute the square root of $x$ in round-towards-zero mode.

Note:

- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- Requires compute capability $\geq$ 2.0.
__device__ double __dsub_rd (double x, double y)
Subtract two floating-point values in round-down mode.

Returns
Returns \( x - y \).

Description
Subtracts two floating-point values \( x \) and \( y \) in round-down (to negative infinity) mode.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.

__device__ double __dsub_rn (double x, double y)
Subtract two floating-point values in round-to-nearest-even mode.

Returns
Returns \( x - y \).

Description
Subtracts two floating-point values \( x \) and \( y \) in round-to-nearest-even mode.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.

__device__ double __dsub_ru (double x, double y)
Subtract two floating-point values in round-up mode.

Returns
Returns \( x - y \).
Description
Subtracts two floating-point values \(x\) and \(y\) in round-up (to positive infinity) mode.

\[
\text{Note:}
\begin{align*}
&\text{For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.} \\
&\text{This operation will never be merged into a single multiply-add instruction.}
\end{align*}
\]

\textbf{__device__ double __dsub_rz (double x, double y)}
Subtract two floating-point values in round-towards-zero mode.

Returns
Returns \(x - y\).

Description
Subtracts two floating-point values \(x\) and \(y\) in round-towards-zero mode.

\[
\text{Note:}
\begin{align*}
&\text{For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.} \\
&\text{This operation will never be merged into a single multiply-add instruction.}
\end{align*}
\]

\textbf{__device__ double __fma_rd (double x, double y, double z)}
Compute \(x \times y + z\) as a single operation in round-down mode.

Returns
Returns the rounded value of \(x \times y + z\) as a single operation.

\[
\begin{align*}
&\text{fmaf} (\pm \infty, \pm 0, z) \text{ returns NaN.} \\
&\text{fmaf} (\pm 0, \pm \infty, z) \text{ returns NaN.} \\
&\text{fmaf}(x, y, -\infty) \text{ returns NaN if } x \times y \text{ is an exact } +\infty \\
&\text{fmaf}(x, y, +\infty) \text{ returns NaN if } x \times y \text{ is an exact } -\infty
\end{align*}
\]
Description
Computes the value of $x \times y + z$ as a single ternary operation, rounding the result once in round-down (to negative infinity) mode.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

__device__ double __fma_rn (double x, double y, double z)
Compute $x \times y + z$ as a single operation in round-to-nearest-even mode.

Returns
Returns the rounded value of $x \times y + z$ as a single operation.

‣ $\text{fma}(\pm \infty, 0, z)$ returns NaN.
‣ $\text{fma}(0, \pm \infty, z)$ returns NaN.
‣ $\text{fma}(x, y, -\infty)$ returns NaN if $x \times y$ is an exact $+\infty$
‣ $\text{fma}(x, y, +\infty)$ returns NaN if $x \times y$ is an exact $-\infty$

Description
Computes the value of $x \times y + z$ as a single ternary operation, rounding the result once in round-to-nearest-even mode.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

__device__ double __fma_ru (double x, double y, double z)
Compute $x \times y + z$ as a single operation in round-up mode.

Returns
Returns the rounded value of $x \times y + z$ as a single operation.
- \( fmaf(\pm\infty, \pm0, z) \) returns NaN.
- \( fmaf(\pm0, \pm\infty, z) \) returns NaN.
- \( fmaf(x, y, -\infty) \) returns NaN if \( x \times y \) is an exact \( +\infty \)
- \( fmaf(x, y, +\infty) \) returns NaN if \( x \times y \) is an exact \( -\infty \)

**Description**

Computes the value of \( x \times y + z \) as a single ternary operation, rounding the result once in round-up (to positive infinity) mode.

**Note:**

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

```c
__device__ double __fma_rz (double x, double y, double z)
```

Compute \( x \times y + z \) as a single operation in round-towards-zero mode.

**Returns**

Returns the rounded value of \( x \times y + z \) as a single operation.

- \( fmaf(\pm\infty, \pm0, z) \) returns NaN.
- \( fmaf(\pm0, \pm\infty, z) \) returns NaN.
- \( fmaf(x, y, -\infty) \) returns NaN if \( x \times y \) is an exact \( +\infty \)
- \( fmaf(x, y, +\infty) \) returns NaN if \( x \times y \) is an exact \( -\infty \)

**Description**

Computes the value of \( x \times y + z \) as a single ternary operation, rounding the result once in round-towards-zero mode.

**Note:**

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
1.10. Integer Intrinsics

This section describes integer intrinsic functions that are only supported in device code. To use these functions you do not need to include any additional header files in your program.

__device__ unsigned int __brev (unsigned int x)
Reverse the bit order of a 32-bit unsigned integer.

Returns
Returns the bit-reversed value of x. i.e. bit N of the return value corresponds to bit 31-N of x.

Description
Reverses the bit order of the 32-bit unsigned integer x.

__device__ unsigned long long int __brevll (unsigned long long int x)
Reverse the bit order of a 64-bit unsigned integer.

Returns
Returns the bit-reversed value of x. i.e. bit N of the return value corresponds to bit 63-N of x.

Description
Reverses the bit order of the 64-bit unsigned integer x.

__device__ unsigned int __byte_perm (unsigned int x, unsigned int y, unsigned int s)
Return selected bytes from two 32-bit unsigned integers.

Returns
Returns a 32-bit integer consisting of four bytes from eight input bytes provided in the two input integers x and y, as specified by a selector, s.

Description
Create 8-byte source

- uint64_t tmp64 = (uint64_t)y << 32 | x;
Extract selector bits

- \( \text{selector0} = (s >> 0) \& 0x7; \)
- \( \text{selector1} = (s >> 4) \& 0x7; \)
- \( \text{selector2} = (s >> 8) \& 0x7; \)
- \( \text{selector3} = (s >> 12) \& 0x7; \)

Return 4 selected bytes from 8-byte source:

- \( \text{res[07:00]} = \text{tmp64[selector0]}; \)
- \( \text{res[15:08]} = \text{tmp64[selector1]}; \)
- \( \text{res[23:16]} = \text{tmp64[selector2]}; \)
- \( \text{res[31:24]} = \text{tmp64[selector3]}; \)

`__device__ int __clz (int x)`

Return the number of consecutive high-order zero bits in a 32-bit integer.

Returns

Returns a value between 0 and 32 inclusive representing the number of zero bits.

Description

Count the number of consecutive leading zero bits, starting at the most significant bit (bit 31) of \( x \).

`__device__ int __clzll (long long int x)`

Count the number of consecutive high-order zero bits in a 64-bit integer.

Returns

Returns a value between 0 and 64 inclusive representing the number of zero bits.

Description

Count the number of consecutive leading zero bits, starting at the most significant bit (bit 63) of \( x \).
__device__ unsigned int __dp2a_hi (ushort2 srcA, uchar4 srcB, unsigned int c)
Two-way unsigned int16 by int8 dot product with unsigned int32 accumulate, taking the upper half of the second input.

Description
Takes two packed 16-bit integers from srcA vector and two packed 8-bit integers from the upper 16 bits of srcB vector, then creates two pairwise 8x16 products and adds them together to an unsigned 32-bit integer c.

__device__ int __dp2a_hi (short2 srcA, char4 srcB, int c)
Two-way signed int16 by int8 dot product with int32 accumulate, taking the upper half of the second input.

Description
Takes two packed 16-bit integers from srcA vector and two packed 8-bit integers from the upper 16 bits of srcB vector, then creates two pairwise 8x16 products and adds them together to a signed 32-bit integer c.

__device__ unsigned int __dp2a_hi (unsigned int srcA, unsigned int srcB, unsigned int c)
Two-way unsigned int16 by int8 dot product with unsigned int32 accumulate, taking the upper half of the second input.

Description
Extracts two packed 16-bit integers from srcA and two packed 8-bit integers from the upper 16 bits of srcB, then creates two pairwise 8x16 products and adds them together to an unsigned 32-bit integer c.

__device__ int __dp2a_hi (int srcA, int srcB, int c)
Two-way signed int16 by int8 dot product with int32 accumulate, taking the upper half of the second input.

Description
Extracts two packed 16-bit integers from srcA and two packed 8-bit integers from the upper 16 bits of srcB, then creates two pairwise 8x16 products and adds them together to a signed 32-bit integer c.
__device__ unsigned int __dp2a_lo (ushort2 srcA, uchar4 srcB, unsigned int c)

Two-way unsigned int16 by int8 dot product with unsigned int32 accumulate, taking the lower half of the second input.

Description
Takes two packed 16-bit integers from srcA vector and two packed 8-bit integers from the lower 16 bits of srcB vector, then creates two pairwise 8x16 products and adds them together to an unsigned 32-bit integer c.

__device__ int __dp2a_lo (short2 srcA, char4 srcB, int c)

Two-way signed int16 by int8 dot product with int32 accumulate, taking the lower half of the second input.

Description
Takes two packed 16-bit integers from srcA vector and two packed 8-bit integers from the lower 16 bits of srcB vector, then creates two pairwise 8x16 products and adds them together to a signed 32-bit integer c.

__device__ unsigned int __dp2a_lo (unsigned int srcA, unsigned int srcB, unsigned int c)

Two-way unsigned int16 by int8 dot product with unsigned int32 accumulate, taking the lower half of the second input.

Description
Extracts two packed 16-bit integers from srcA and two packed 8-bit integers from the lower 16 bits of srcB, then creates two pairwise 8x16 products and adds them together to an unsigned 32-bit integer c.

__device__ int __dp2a_lo (int srcA, int srcB, int c)

Two-way signed int16 by int8 dot product with int32 accumulate, taking the lower half of the second input.

Description
Extracts two packed 16-bit integers from srcA and two packed 8-bit integers from the lower 16 bits of srcB, then creates two pairwise 8x16 products and adds them together to a signed 32-bit integer c.
__device__ unsigned int __dp4a (uchar4 srcA, uchar4 srcB, unsigned int c)
Four-way unsigned int8 dot product with unsigned int32 accumulate.

Description
Takes four pairs of packed byte-sized integers from srcA and srcB vectors, then creates four pairwise products and adds them together to an unsigned 32-bit integer c.

__device__ int __dp4a (char4 srcA, char4 srcB, int c)
Four-way signed int8 dot product with int32 accumulate.

Description
Takes four pairs of packed byte-sized integers from srcA and srcB vectors, then creates four pairwise products and adds them together to a signed 32-bit integer c.

__device__ unsigned int __dp4a (unsigned int srcA, unsigned int srcB, unsigned int c)
Four-way unsigned int8 dot product with unsigned int32 accumulate.

Description
Extracts four pairs of packed byte-sized integers from srcA and srcB, then creates four pairwise products and adds them together to an unsigned 32-bit integer c.

__device__ int __dp4a (int srcA, int srcB, int c)
Four-way signed int8 dot product with int32 accumulate.

Description
Extracts four pairs of packed byte-sized integers from srcA and srcB, then creates four pairwise products and adds them together to a signed 32-bit integer c.

__device__ int __ffs (int x)
Find the position of the least significant bit set to 1 in a 32-bit integer.

Returns
Returns a value between 0 and 32 inclusive representing the position of the first bit set.

- __ffs(0) returns 0.
Description
Find the position of the first (least significant) bit set to 1 in \( x \), where the least significant bit position is 1.

\[
\_\_d\_e\_v\_i\_c\_e\_r\_ \_i\_n\_t \_\_f\_f\_s\_l\_l \ (l\_o\_n\_g \ l\_o\_n\_g \ i\_n\_t \ x)
\]
Find the position of the least significant bit set to 1 in a 64-bit integer.

Returns
Returns a value between 0 and 64 inclusive representing the position of the first bit set.

- \_\_ffs\_l\_l(0) returns 0.

Description
Find the position of the first (least significant) bit set to 1 in \( x \), where the least significant bit position is 1.

\[
\_\_d\_e\_v\_i\_c\_e\_r\_ \_i\_n\_t \_\_f\_n\_s\_n\_ (u\_n\_s\_n\_i\_g\_s \ m\_a\_s\_k, \ u\_n\_s\_n\_i\_g\_s \ b\_a\_s\_e, \ i\_n\_t \ o\_f\_s\_e\_t\_t)
\]
Find the position of the n-th set to 1 bit in a 32-bit integer.

Returns
Returns a value between 0 and 32 inclusive representing the position of the n-th set bit.

- parameter base must be \( \leq 31 \), otherwise behavior is undefined.

Description
Given a 32-bit value mask and an integer value base (between 0 and 31), find the n-th (given by offset) set bit in mask from the base bit. If not found, return 0xFFFFFFFF.

See also [https://docs.nvidia.com/cuda/parallel-thread-execution/index.html#integer-arithmetic-instructions-fns](https://docs.nvidia.com/cuda/parallel-thread-execution/index.html#integer-arithmetic-instructions-fns) for more information.

\[
\_\_d\_e\_v\_i\_c\_e\_r\_ \_i\_n\_t \_\_f\_u\_n\_n\_e\_l\_s\_h\_i\_f\_s\_t\_l\_l \ (u\_n\_s\_n\_i\_g\_s \ i\_n\_t \ l\_o, \ u\_n\_s\_n\_i\_g\_s \ i\_n\_t \ h\_i, \ u\_n\_s\_n\_i\_g\_s \ i\_n\_t \ s\_h\_i\_f\_t)
\]
Concatenate hi : lo, shift left by shift & 31 bits, return the most significant 32 bits.

Returns
Returns the most significant 32 bits of the shifted 64-bit value.
Description
Shift the 64-bit value formed by concatenating argument lo and hi left by the amount specified by the argument shift. Argument lo holds bits 31:0 and argument hi holds bits 63:32 of the 64-bit source value. The source is shifted left by the wrapped value of shift (shift & 31). The most significant 32-bits of the result are returned.

__device__ unsigned int __funnelshift_lc (unsigned int lo, unsigned int hi, unsigned int shift)
Concatenate hi:lo, shift left by min(shift, 32) bits, return the most significant 32 bits.

Returns
Returns the most significant 32 bits of the shifted 64-bit value.

Description
Shift the 64-bit value formed by concatenating argument lo and hi right by the amount specified by the argument shift. Argument lo holds bits 31:0 and argument hi holds bits 63:32 of the 64-bit source value. The source is shifted right by the wrapped value of shift (shift & 31). The least significant 32-bits of the result are returned.

__device__ unsigned int __funnelshift_r (unsigned int lo, unsigned int hi, unsigned int shift)
Concatenate hi:lo, shift right by shift & 31 bits, return the least significant 32 bits.

Returns
Returns the least significant 32 bits of the shifted 64-bit value.
__device__ unsigned int __funnelshift_rc (unsigned int lo, unsigned int hi, unsigned int shift)

Concatenate $hi : lo$, shift right by $\min(shift, 32)$ bits, return the least significant 32 bits.

Returns
Returns the least significant 32 bits of the shifted 64-bit value.

Description
Shift the 64-bit value formed by concatenating argument $lo$ and $hi$ right by the amount specified by the argument $shift$. Argument $lo$ holds bits 31:0 and argument $hi$ holds bits 63:32 of the 64-bit source value. The source is shifted right by the clamped value of $shift (\min(shift, 32))$. The least significant 32-bits of the result are returned.

__device__ int __hadd (int x, int y)

Compute average of signed input arguments, avoiding overflow in the intermediate sum.

Returns
Returns a signed integer value representing the signed average value of the two inputs.

Description
Compute average of signed input arguments $x$ and $y$ as $(x + y) >> 1$, avoiding overflow in the intermediate sum.

__device__ int __mul24 (int x, int y)

Calculate the least significant 32 bits of the product of the least significant 24 bits of two integers.

Returns
Returns the least significant 32 bits of the product $x \times y$.

Description
Calculate the least significant 32 bits of the product of the least significant 24 bits of $x$ and $y$. The high order 8 bits of $x$ and $y$ are ignored.
__device__ long long int __mul64hi (long long int x, long long int y)
Calculate the most significant 64 bits of the product of the two 64-bit integers.

Returns
Returns the most significant 64 bits of the product $x \times y$.

Description
Calculate the most significant 64 bits of the 128-bit product $x \times y$, where $x$ and $y$ are 64-bit integers.

__device__ int __mulhi (int x, int y)
Calculate the most significant 32 bits of the product of the two 32-bit integers.

Returns
Returns the most significant 32 bits of the product $x \times y$.

Description
Calculate the most significant 32 bits of the 64-bit product $x \times y$, where $x$ and $y$ are 32-bit integers.

__device__ int __popc (unsigned int x)
Count the number of bits that are set to 1 in a 32-bit integer.

Returns
Returns a value between 0 and 32 inclusive representing the number of set bits.

Description
Count the number of bits that are set to 1 in $x$.

__device__ int __popcll (unsigned long long int x)
Count the number of bits that are set to 1 in a 64-bit integer.

Returns
Returns a value between 0 and 64 inclusive representing the number of set bits.
Description
Count the number of bits that are set to 1 in \( x \).

\textbf{\_\_device\_\_ int \_\_rhadd (int \_\_x, int \_\_y)}
Compute rounded average of signed input arguments, avoiding overflow in the intermediate sum.

\textbf{Returns}
Returns a signed integer value representing the signed rounded average value of the two inputs.

Description
Compute average of signed input arguments \( x \) and \( y \) as \((x + y + 1) \gg 1\), avoiding overflow in the intermediate sum.

\textbf{\_\_device\_\_ unsigned int \_\_sad (int \_\_x, int \_\_y, unsigned int \_\_z)}
Calculate \( |x - y| + z \), the sum of absolute difference.

\textbf{Returns}
Returns \( |x - y| + z \).

Description
Calculate \( |x - y| + z \), the 32-bit sum of the third argument \( z \) plus and the absolute value of the difference between the first argument, \( x \), and second argument, \( y \). Inputs \( x \) and \( y \) are signed 32-bit integers, input \( z \) is a 32-bit unsigned integer.

\textbf{\_\_device\_\_ unsigned int \_\_uhadd (unsigned int \_\_x, unsigned int \_\_y)}
Compute average of unsigned input arguments, avoiding overflow in the intermediate sum.

\textbf{Returns}
Returns an unsigned integer value representing the unsigned average value of the two inputs.
Description
Compute average of unsigned input arguments x and y as \((x + y) \gg 1\), avoiding overflow in the intermediate sum.

__device__ unsigned int __umul24 (unsigned int x, unsigned int y)
Calculate the least significant 32 bits of the product of the least significant 24 bits of two unsigned integers.

Returns
Returns the least significant 32 bits of the product \(x \times y\).

Description
Calculate the least significant 32 bits of the product of the least significant 24 bits of \(x\) and \(y\). The high order 8 bits of \(x\) and \(y\) are ignored.

__device__ unsigned long long int __umul64hi (unsigned long long int x, unsigned long long int y)
Calculate the most significant 64 bits of the product of the two 64 unsigned bit integers.

Returns
Returns the most significant 64 bits of the product \(x \times y\).

Description
Calculate the most significant 64 bits of the 128-bit product \(x \times y\), where \(x\) and \(y\) are 64-bit unsigned integers.

__device__ unsigned int __umulhi (unsigned int x, unsigned int y)
Calculate the most significant 32 bits of the product of the two 32-bit unsigned integers.

Returns
Returns the most significant 32 bits of the product \(x \times y\).
Description
Calculate the most significant 32 bits of the 64-bit product \( x \times y \), where \( x \) and \( y \) are 32-bit unsigned integers.

\[
\_\_d e v i c e \_\_ u n s i g n e d \ i n t \ _\_u r h a d d (u n s i g n e d \ i n t \ x, \ u n s i g n e d \ i n t \ y)
\]
Compute rounded average of unsigned input arguments, avoiding overflow in the intermediate sum.

Returns
Returns an unsigned integer value representing the unsigned rounded average value of the two inputs.

Description
Compute average of unsigned input arguments \( x \) and \( y \) as \((x + y + 1) \gg 1\), avoiding overflow in the intermediate sum.

\[
\_\_d e v i c e \_\_ u n s i g n e d \ i n t \ _\_u s a d (u n s i g n e d \ i n t \ x, \ u n s i g n e d \ i n t \ y, \ u n s i g n e d \ i n t \ z)
\]
Calculate \(|x - y| + z\), the sum of absolute difference.

Returns
Returns \(|x - y| + z\).

Description
Calculate \(|x - y| + z\), the 32-bit sum of the third argument \( z \) plus and the absolute value of the difference between the first argument, \( x \), and second argument, \( y \).
Inputs \( x \), \( y \), and \( z \) are unsigned 32-bit integers.

1.11. Type Casting Intrinsics
This section describes type casting intrinsic functions that are only supported in device code. To use these functions you do not need to include any additional header files in your program.
__device__ float __double2float_rd (double x)
Convert a double to a float in round-down mode.

Returns
Returns converted value.

Description
Convert the double-precision floating-point value \( x \) to a single-precision floating-point value in round-down (to negative infinity) mode.

__device__ float __double2float_rn (double x)
Convert a double to a float in round-to-nearest-even mode.

Returns
Returns converted value.

Description
Convert the double-precision floating-point value \( x \) to a single-precision floating-point value in round-to-nearest-even mode.

__device__ float __double2float_ru (double x)
Convert a double to a float in round-up mode.

Returns
Returns converted value.

Description
Convert the double-precision floating-point value \( x \) to a single-precision floating-point value in round-up (to positive infinity) mode.

__device__ float __double2float_rz (double x)
Convert a double to a float in round-towards-zero mode.

Returns
Returns converted value.
**Description**
Convert the double-precision floating-point value \( x \) to a single-precision floating-point value in round-towards-zero mode.

\[ \text{__device__ int __double2hiint (double x)} \]
Reinterpret high 32 bits in a double as a signed integer.

**Returns**
Returns reinterpreted value.

**Description**
Reinterpret the high 32 bits in the double-precision floating-point value \( x \) as a signed integer.

\[ \text{__device__ int __double2int_rd (double x)} \]
Convert a double to a signed int in round-down mode.

**Returns**
Returns converted value.

**Description**
Convert the double-precision floating-point value \( x \) to a signed integer value in round-down (to negative infinity) mode.

\[ \text{__device__ int __double2int_rn (double x)} \]
Convert a double to a signed int in round-to-nearest-even mode.

**Returns**
Returns converted value.

**Description**
Convert the double-precision floating-point value \( x \) to a signed integer value in round-to-nearest-even mode.
__device__ int __double2int_ru (double x)
Convert a double to a signed int in round-up mode.

Returns
Returns converted value.

Description
Convert the double-precision floating-point value \( x \) to a signed integer value in round-up (to positive infinity) mode.

__device__ int __double2int_rz (double x)
Convert a double to a signed int in round-towards-zero mode.

Returns
Returns converted value.

Description
Convert the double-precision floating-point value \( x \) to a signed integer value in round-towards-zero mode.

__device__ long long int __double2ll_rd (double x)
Convert a double to a signed 64-bit int in round-down mode.

Returns
Returns converted value.

Description
Convert the double-precision floating-point value \( x \) to a signed 64-bit integer value in round-down (to negative infinity) mode.

__device__ long long int __double2ll_rn (double x)
Convert a double to a signed 64-bit int in round-to-nearest-even mode.

Returns
Returns converted value.
Description
Convert the double-precision floating-point value \(x\) to a signed 64-bit integer value in round-to-nearest-even mode.

```c
__device__ long long int __double2ll_ru (double x)
```
Convert a double to a signed 64-bit int in round-up mode.

Returns
Returns converted value.

Description
Convert the double-precision floating-point value \(x\) to a signed 64-bit integer value in round-up (to positive infinity) mode.

```c
__device__ long long int __double2ll_rz (double x)
```
Convert a double to a signed 64-bit int in round-towards-zero mode.

Returns
Returns converted value.

Description
Convert the double-precision floating-point value \(x\) to a signed 64-bit integer value in round-towards-zero mode.

```c
__device__ int __double2loint (double x)
```
Reinterpret low 32 bits in a double as a signed integer.

Returns
Returns reinterpreted value.

Description
Reinterpret the low 32 bits in the double-precision floating-point value \(x\) as a signed integer.
__device__ unsigned int __double2uint_rd (double x)
Convert a double to an unsigned int in round-down mode.

Returns
Returns converted value.

Description
Convert the double-precision floating-point value \( x \) to an unsigned integer value in round-down (to negative infinity) mode.

__device__ unsigned int __double2uint_rn (double x)
Convert a double to an unsigned int in round-to-nearest-even mode.

Returns
Returns converted value.

Description
Convert the double-precision floating-point value \( x \) to an unsigned integer value in round-to-nearest-even mode.

__device__ unsigned int __double2uint_ru (double x)
Convert a double to an unsigned int in round-up mode.

Returns
Returns converted value.

Description
Convert the double-precision floating-point value \( x \) to an unsigned integer value in round-up (to positive infinity) mode.

__device__ unsigned int __double2uint_rz (double x)
Convert a double to an unsigned int in round-towards-zero mode.

Returns
Returns converted value.
**Description**
Convert the double-precision floating-point value $x$ to an unsigned integer value in round-towards-zero mode.

```c
__device__ unsigned long long int __double2ull_rd (double x)
```
Convert a double to an unsigned 64-bit int in round-down mode.

**Returns**
Returns converted value.

**Description**
Convert the double-precision floating-point value $x$ to an unsigned 64-bit integer value in round-down (to negative infinity) mode.

```c
__device__ unsigned long long int __double2ull_rn (double x)
```
Convert a double to an unsigned 64-bit int in round-to-nearest-even mode.

**Returns**
Returns converted value.

**Description**
Convert the double-precision floating-point value $x$ to an unsigned 64-bit integer value in round-to-nearest-even mode.

```c
__device__ unsigned long long int __double2ull_ru (double x)
```
Convert a double to an unsigned 64-bit int in round-up mode.

**Returns**
Returns converted value.

**Description**
Convert the double-precision floating-point value $x$ to an unsigned 64-bit integer value in round-up (to positive infinity) mode.
__device__ unsigned long long int __double2ull_rz (double x)
Convert a double to an unsigned 64-bit int in round-towards-zero mode.

Returns
Returns converted value.

Description
Convert the double-precision floating-point value x to an unsigned 64-bit integer value in round-towards-zero mode.

__device__ long long int __double_as_longlong (double x)
Reinterpret bits in a double as a 64-bit signed integer.

Returns
Returns reinterpreted value.

Description
Reinterpret the bits in the double-precision floating-point value x as a signed 64-bit integer.

__device__ int __float2int_rd (float x)
Convert a float to a signed integer in round-down mode.

Returns
Returns converted value.

Description
Convert the single-precision floating-point value x to a signed integer in round-down (to negative infinity) mode.

__device__ int __float2int_rn (float x)
Convert a float to a signed integer in round-to-nearest-even mode.

Returns
Returns converted value.
Description
Convert the single-precision floating-point value $x$ to a signed integer in round-to-nearest-even mode.

__device__ int __float2int_ru (float)
Convert a float to a signed integer in round-up mode.

Returns
Returns converted value.

Description
Convert the single-precision floating-point value $x$ to a signed integer in round-up (to positive infinity) mode.

__device__ int __float2int_rz (float x)
Convert a float to a signed integer in round-towards-zero mode.

Returns
Returns converted value.

Description
Convert the single-precision floating-point value $x$ to a signed integer in round-towards-zero mode.

__device__ long long int __float2ll_rd (float x)
Convert a float to a signed 64-bit integer in round-down mode.

Returns
Returns converted value.

Description
Convert the single-precision floating-point value $x$ to a signed 64-bit integer in round-down (to negative infinity) mode.
__device__ long long int __float2ll_rn (float x)
Convert a float to a signed 64-bit integer in round-to-nearest-even mode.

Returns
Returns converted value.

Description
Convert the single-precision floating-point value \( x \) to a signed 64-bit integer in round-to-nearest-even mode.

__device__ long long int __float2ll_ru (float x)
Convert a float to a signed 64-bit integer in round-up mode.

Returns
Returns converted value.

Description
Convert the single-precision floating-point value \( x \) to a signed 64-bit integer in round-up (to positive infinity) mode.

__device__ long long int __float2ll_rz (float x)
Convert a float to a signed 64-bit integer in round-towards-zero mode.

Returns
Returns converted value.

Description
Convert the single-precision floating-point value \( x \) to a signed 64-bit integer in round-towards-zero mode.

__device__ unsigned int __float2uint_rd (float x)
Convert a float to an unsigned integer in round-down mode.

Returns
Returns converted value.
Description
Convert the single-precision floating-point value \( x \) to an unsigned integer in round-down (to negative infinity) mode.

```
__device__ unsigned int __float2uint_rn (float x)
```
Convert a float to an unsigned integer in round-to-nearest-even mode.

Returns
Returns converted value.

Description
Convert the single-precision floating-point value \( x \) to an unsigned integer in round-to-nearest-even mode.

```
__device__ unsigned int __float2uint_ru (float x)
```
Convert a float to an unsigned integer in round-up mode.

Returns
Returns converted value.

Description
Convert the single-precision floating-point value \( x \) to an unsigned integer in round-up (to positive infinity) mode.

```
__device__ unsigned int __float2uint_rz (float x)
```
Convert a float to an unsigned integer in round-towards-zero mode.

Returns
Returns converted value.

Description
Convert the single-precision floating-point value \( x \) to an unsigned integer in round-towards-zero mode.
__device__ unsigned long long int __float2ull_rd (float x)
Convert a float to an unsigned 64-bit integer in round-down mode.

Returns
Returns converted value.

Description
Convert the single-precision floating-point value x to an unsigned 64-bit integer in round-down (to negative infinity) mode.

__device__ unsigned long long int __float2ull_rn (float x)
Convert a float to an unsigned 64-bit integer in round-to-nearest-even mode.

Returns
Returns converted value.

Description
Convert the single-precision floating-point value x to an unsigned 64-bit integer in round-to-nearest-even mode.

__device__ unsigned long long int __float2ull_ru (float x)
Convert a float to an unsigned 64-bit integer in round-up mode.

Returns
Returns converted value.

Description
Convert the single-precision floating-point value x to an unsigned 64-bit integer in round-up (to positive infinity) mode.
__device__ unsigned long long int __float2ull_rz (float x)
Convert a float to an unsigned 64-bit integer in round-towards-zero mode.

Returns
Returns converted value.

Description
Convert the single-precision floating-point value \( x \) to an unsigned 64-bit integer in round-towards-zero mode.

__device__ int __float_as_int (float x)
Reinterpret bits in a float as a signed integer.

Returns
Returns reinterpreted value.

Description
Reinterpret the bits in the single-precision floating-point value \( x \) as a signed integer.

__device__ unsigned int __float_as_uint (float x)
Reinterpret bits in a float as an unsigned integer.

Returns
Returns reinterpreted value.

Description
Reinterpret the bits in the single-precision floating-point value \( x \) as an unsigned integer.

__device__ double __hiloint2double (int hi, int lo)
Reinterpret high and low 32-bit integer values as a double.

Returns
Returns reinterpreted value.
Description
Reinterpret the integer value of \( hi \) as the high 32 bits of a double-precision floating-point value and the integer value of \( lo \) as the low 32 bits of the same double-precision floating-point value.

\[
\text{\texttt{\_device\_ double \_int2double\_rn (int x)}}
\]
Convert a signed int to a double.

Returns
Returns converted value.

Description
Convert the signed integer value \( x \) to a double-precision floating-point value.

\[
\text{\texttt{\_device\_ float \_int2float\_rd (int x)}}
\]
Convert a signed integer to a float in round-down mode.

Returns
Returns converted value.

Description
Convert the signed integer value \( x \) to a single-precision floating-point value in round-down (to negative infinity) mode.

\[
\text{\texttt{\_device\_ float \_int2float\_rn (int x)}}
\]
Convert a signed integer to a float in round-to-nearest-even mode.

Returns
Returns converted value.

Description
Convert the signed integer value \( x \) to a single-precision floating-point value in round-to-nearest-even mode.
__device__ float __int2float_ru (int x)
Convert a signed integer to a float in round-up mode.

Returns
Returns converted value.

Description
Convert the signed integer value \( x \) to a single-precision floating-point value in round-up (to positive infinity) mode.

__device__ float __int2float_rz (int x)
Convert a signed integer to a float in round-towards-zero mode.

Returns
Returns converted value.

Description
Convert the signed integer value \( x \) to a single-precision floating-point value in round-towards-zero mode.

__device__ float __int_as_float (int x)
Reinterpret bits in an integer as a float.

Returns
Returns reinterpreted value.

Description
Reinterpret the bits in the signed integer value \( x \) as a single-precision floating-point value.

__device__ double __ll2double_rd (long long int x)
Convert a signed 64-bit int to a double in round-down mode.

Returns
Returns converted value.
Description
Convert the signed 64-bit integer value \( x \) to a double-precision floating-point value in round-down (to negative infinity) mode.

```c
__device__ double __ll2double_rn (long long int x)
```
Convert a signed 64-bit int to a double in round-to-nearest-even mode.

Returns
Returns converted value.

Description
Convert the signed 64-bit integer value \( x \) to a double-precision floating-point value in round-to-nearest-even mode.

```c
__device__ double __ll2double_ru (long long int x)
```
Convert a signed 64-bit int to a double in round-up mode.

Returns
Returns converted value.

Description
Convert the signed 64-bit integer value \( x \) to a double-precision floating-point value in round-up (to positive infinity) mode.

```c
__device__ double __ll2double_rz (long long int x)
```
Convert a signed 64-bit int to a double in round-towards-zero mode.

Returns
Returns converted value.
__device__ float __ll2float_rd (long long int x)
Convert a signed integer to a float in round-down mode.

Returns
Returns converted value.

Description
Convert the signed integer value x to a single-precision floating-point value in round-down [to negative infinity] mode.

__device__ float __ll2float_rn (long long int x)
Convert a signed 64-bit integer to a float in round-to-nearest-even mode.

Returns
Returns converted value.

Description
Convert the signed 64-bit integer value x to a single-precision floating-point value in round-to-nearest-even mode.

__device__ float __ll2float_ru (long long int x)
Convert a signed integer to a float in round-up mode.

Returns
Returns converted value.

Description
Convert the signed integer value x to a single-precision floating-point value in round-up [to positive infinity] mode.

__device__ float __ll2float_rz (long long int x)
Convert a signed integer to a float in round-towards-zero mode.

Returns
Returns converted value.
Description
Convert the signed integer value x to a single-precision floating-point value in round-towards-zero mode.

__device__ double __longlong_as_double (long long int x)
Reinterpret bits in a 64-bit signed integer as a double.

Returns
Returns reinterpreted value.

Description
Reinterpret the bits in the 64-bit signed integer value x as a double-precision floating-point value.

__device__ double __uint2double_rn (unsigned int x)
Convert an unsigned int to a double.

Returns
Returns converted value.

Description
Convert the unsigned integer value x to a double-precision floating-point value.

__device__ float __uint2float_rd (unsigned int x)
Convert an unsigned integer to a float in round-down mode.

Returns
Returns converted value.

Description
Convert the unsigned integer value x to a single-precision floating-point value in round-down (to negative infinity) mode.
__device__ float __uint2float_rn (unsigned int x)
Convert an unsigned integer to a float in round-to-nearest-even mode.

Returns
Returns converted value.

Description
Convert the unsigned integer value \( x \) to a single-precision floating-point value in round-to-nearest-even mode.

__device__ float __uint2float_ru (unsigned int x)
Convert an unsigned integer to a float in round-up mode.

Returns
Returns converted value.

Description
Convert the unsigned integer value \( x \) to a single-precision floating-point value in round-up (to positive infinity) mode.

__device__ float __uint2float_rz (unsigned int x)
Convert an unsigned integer to a float in round-towards-zero mode.

Returns
Returns converted value.

Description
Convert the unsigned integer value \( x \) to a single-precision floating-point value in round-towards-zero mode.

__device__ float __uint_as_float (unsigned int x)
Reinterpret bits in an unsigned integer as a float.

Returns
Returns reinterpreted value.
Description
Reinterpret the bits in the unsigned integer value \( x \) as a single-precision floating-point value.

`__device__ double __ull2double_rd (unsigned long long int x)`
Convert an unsigned 64-bit int to a double in round-down mode.

Returns
Returns converted value.

Description
Convert the unsigned 64-bit integer value \( x \) to a double-precision floating-point value in round-down (to negative infinity) mode.

`__device__ double __ull2double_rn (unsigned long long int x)`
Convert an unsigned 64-bit int to a double in round-to-nearest-even mode.

Returns
Returns converted value.

Description
Convert the unsigned 64-bit integer value \( x \) to a double-precision floating-point value in round-to-nearest-even mode.

`__device__ double __ull2double_ru (unsigned long long int x)`
Convert an unsigned 64-bit int to a double in round-up mode.

Returns
Returns converted value.

Description
Convert the unsigned 64-bit integer value \( x \) to a double-precision floating-point value in round-up (to positive infinity) mode.
\texttt{\_device\_ double \_ull2double\_rz (unsigned long long int x)}

Convert an unsigned 64-bit int to a double in round-towards-zero mode.

\textbf{Returns}

Returns converted value.

\textbf{Description}

Convert the unsigned 64-bit integer value \texttt{x} to a double-precision floating-point value in round-towards-zero mode.

\texttt{\_device\_ float \_ull2float\_rd (unsigned long long int x)}

Convert an unsigned integer to a float in round-down mode.

\textbf{Returns}

Returns converted value.

\textbf{Description}

Convert the unsigned integer value \texttt{x} to a single-precision floating-point value in round-down (to negative infinity) mode.

\texttt{\_device\_ float \_ull2float\_rn (unsigned long long int x)}

Convert an unsigned integer to a float in round-to-nearest-even mode.

\textbf{Returns}

Returns converted value.

\textbf{Description}

Convert the unsigned integer value \texttt{x} to a single-precision floating-point value in round-to-nearest-even mode.
__device__ float __ull2float_ru (unsigned long long int x)
Convert an unsigned integer to a float in round-up mode.

Returns
Returns converted value.

Description
Convert the unsigned integer value x to a single-precision floating-point value in round-up mode.

__device__ float __ull2float_rz (unsigned long long int x)
Convert an unsigned integer to a float in round-towards-zero mode.

Returns
Returns converted value.

Description
Convert the unsigned integer value x to a single-precision floating-point value in round-towards-zero mode.

1.12. SIMD Intrinsics

This section describes SIMD intrinsic functions that are only supported in device code. To use these functions you do not need to include any additional header files in your program.

__device__ unsigned int __vabs2 (unsigned int a)
Computes per-halfword absolute value.

Returns
Returns computed value.

Description
Splits 4 bytes of argument into 2 parts, each consisting of 2 bytes, then computes absolute value for each of parts. Partial results are recombined and returned as unsigned int.
__device__ unsigned int __vabs4 (unsigned int a)
Computes per-byte absolute value.

Returns
Returns computed value.

Description
Splits argument by bytes. Computes absolute value of each byte. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vabsdiffs2 (unsigned int a, unsigned int b)
Computes per-halfword sum of absolute difference of signed integer.

Returns
Returns computed value.

Description
Splits 4 bytes of each into 2 parts, each consisting of 2 bytes. For corresponding parts function computes absolute difference. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vabsdiffs4 (unsigned int a, unsigned int b)
Computes per-byte absolute difference of signed integer.

Returns
Returns computed value.

Description
Splits 4 bytes of each into 4 parts, each consisting of 1 byte. For corresponding parts function computes absolute difference. Partial results are recombined and returned as unsigned int.
__device__ unsigned int __vabsdiffu2 (unsigned int a, unsigned int b)
Performs per-halfword absolute difference of unsigned integer computation: |a - b|.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function computes absolute difference. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vabsdiffu4 (unsigned int a, unsigned int b)
Computes per-byte absolute difference of unsigned integer.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function computes absolute difference. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vabssss2 (unsigned int a)
Computes per-halfword absolute value with signed saturation.

Returns
Returns computed value.

Description
Splits 4 bytes of argument into 2 parts, each consisting of 2 bytes, then computes absolute value with signed saturation for each of parts. Partial results are recombined and returned as unsigned int.
__device__ unsigned int __vabsss4 (unsigned int a)
Computes per-byte absolute value with signed saturation.

Returns
Returns computed value.

Description
Splits 4 bytes of argument into 4 parts, each consisting of 1 byte, then computes absolute value with signed saturation for each of parts. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vadd2 (unsigned int a, unsigned int b)
Performs per-halfword (un)signed addition, with wrap-around: a + b.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes, then performs unsigned addition on corresponding parts. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vadd4 (unsigned int a, unsigned int b)
Performs per-byte (un)signed addition.

Returns
Returns computed value.

Description
Splits 'a' into 4 bytes, then performs unsigned addition on each of these bytes with the corresponding byte from 'b', ignoring overflow. Partial results are recombined and returned as unsigned int.
__device__ unsigned int __vaddss2 (unsigned int a, unsigned int b)
Performs per-halfword addition with signed saturation.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes, then performs addition with signed saturation on corresponding parts. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vaddss4 (unsigned int a, unsigned int b)
Performs per-byte addition with signed saturation.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte, then performs addition with signed saturation on corresponding parts. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vaddus2 (unsigned int a, unsigned int b)
Performs per-halfword addition with unsigned saturation.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes, then performs addition with unsigned saturation on corresponding parts.
__device__ unsigned int __vaddus4 (unsigned int a, unsigned int b)
Performs per-byte addition with unsigned saturation.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte, then performs addition with unsigned saturation on corresponding parts.

__device__ unsigned int __vavgs2 (unsigned int a, unsigned int b)
Performs per-halfword signed rounded average computation.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes, then computes signed rounded average of corresponding parts. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vavgs4 (unsigned int a, unsigned int b)
Computes per-byte signed rounded average.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte, then computes signed rounded average of corresponding parts. Partial results are recombined and returned as unsigned int.
__device__ unsigned int __vavgu2 (unsigned int a, unsigned int b)
Performs per-halfword unsigned rounded average computation.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes, then computes unsigned rounded average of corresponding parts. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vavgu4 (unsigned int a, unsigned int b)
Performs per-byte unsigned rounded average.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. then computes unsigned rounded average of corresponding parts. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vcmpeq2 (unsigned int a, unsigned int b)
Performs per-halfword (un)signed comparison.

Returns
Returns 0xffff computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts result is ffff if they are equal, and 0000 otherwise. For example __vcmpeq2(0x1234aba5, 0x1234aba6) returns 0xffff0000.
__device__ unsigned int __vcmpeq4 (unsigned int a, unsigned int b)
Performs per-byte (un)signed comparison.

Returns
Returns 0xff if a = b, else returns 0.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts result is ff if they are equal, and 00 otherwise. For example __vcmpeq4(0x1234aba5, 0x1234aba6) returns 0xffffff00.

__device__ unsigned int __vcmpges2 (unsigned int a, unsigned int b)
Performs per-halfword signed comparison: a >= b ? 0xffff : 0.

Returns
Returns 0xffff if a >= b, else returns 0.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts result is ffff if 'a' part >= 'b' part, and 0000 otherwise. For example __vcmpges2(0x1234aba5, 0x1234aba6) returns 0xffff0000.

__device__ unsigned int __vcmpges4 (unsigned int a, unsigned int b)
Performs per-byte signed comparison.

Returns
Returns 0xff if a >= b, else returns 0.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts result is ff if 'a' part >= 'b' part, and 00 otherwise. For example __vcmpges4(0x1234aba5, 0x1234aba6) returns 0xffffff00.
`__device__ unsigned int __vcmpgeu2 (unsigned int a, unsigned int b)`
Performs per-halfword unsigned comparison: \( a \geq b \ ? \ 0xffff \ : \ 0 \).

**Returns**
Returns 0xffff if \( a \geq b \), else returns 0.

**Description**
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts result is ffff if ‘a’ part >= ‘b’ part, and 0000 otherwise. For example `__vcmpgeu2(0x1234aba5, 0x1234aba6)` returns 0xffff0000.

`__device__ unsigned int __vcmpgeu4 (unsigned int a, unsigned int b)`
Performs per-byte unsigned comparison.

**Returns**
Returns 0xff if \( a = b \), else returns 0.

**Description**
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts result is ff if ‘a’ part >= ‘b’ part, and 00 otherwise. For example `__vcmpgeu4(0x1234aba5, 0x1234aba6)` returns 0xffffff00.

`__device__ unsigned int __vcmpgts2 (unsigned int a, unsigned int b)`
Performs per-halfword signed comparison: \( a > b \ ? \ 0xffff \ : \ 0 \).

**Returns**
Returns 0xffff if \( a > b \), else returns 0.

**Description**
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts result is ffff if ‘a’ part > ‘b’ part, and 0000 otherwise. For example `__vcmpgts2(0x1234aba5, 0x1234aba6)` returns 0x00000000.
__device__ unsigned int __vcmpgts4 (unsigned int a, unsigned int b)
Performs per-byte signed comparison.

Returns
Returns 0xff if a > b, else returns 0.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts result is ff if ‘a’ part > ‘b’ part, and 00 otherwise. For example __vcmpgts4(0x1234aba5, 0x1234aba6) returns 0x00000000.

__device__ unsigned int __vcmpgtu2 (unsigned int a, unsigned int b)
Performs per-halfword unsigned comparison: a > b ? 0xffff : 0.

Returns
Returns 0xffff if a > b, else returns 0.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts result is ffff if ‘a’ part > ‘b’ part, and 0000 otherwise. For example __vcmpgtu2(0x1234aba5, 0x1234aba6) returns 0x00000000.

__device__ unsigned int __vcmpgtu4 (unsigned int a, unsigned int b)
Performs per-byte unsigned comparison.

Returns
Returns 0xff if a > b, else returns 0.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts result is ff if ‘a’ part > ‘b’ part, and 00 otherwise. For example __vcmpgtu4(0x1234aba5, 0x1234aba6) returns 0x00000000.
__device__ unsigned int __vcmples2 (unsigned int a, unsigned int b)
Performs per-halfword signed comparison: a <= b ? 0xffff : 0.

Returns
Returns 0xffff if a <= b, else returns 0.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts result is ffff if 'a' part <= 'b' part, and 0000 otherwise. For example __vcmples2(0x1234aba5, 0x1234aba6) returns 0xffffffff.

__device__ unsigned int __vcmples4 (unsigned int a, unsigned int b)
Performs per-byte signed comparison.

Returns
Returns 0xff if a <= b, else returns 0.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts result is ff if 'a' part <= 'b' part, and 00 otherwise. For example __vcmples4(0x1234aba5, 0x1234aba6) returns 0xffffffff.

__device__ unsigned int __vcmpleu2 (unsigned int a, unsigned int b)
Performs per-halfword unsigned comparison: a <= b ? 0xffff : 0.

Returns
Returns 0xffff if a <= b, else returns 0.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts result is ffff if 'a' part <= 'b' part, and 0000 otherwise. For example __vcmpleu2(0x1234aba5, 0x1234aba6) returns 0xffffffff.
__device__ unsigned int __vcmpleu4 (unsigned int a, unsigned int b)
Performs per-byte unsigned comparison.

Returns
Returns 0xff if \( a \leq b \), else returns 0.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts result is ff if ‘a’ part \( \leq \) ‘b’ part, and 00 otherwise. For example __vcmpleu4(0x1234aba5, 0x1234aba6) returns 0xffffffff.

__device__ unsigned int __vcmlts2 (unsigned int a, unsigned int b)
Performs per-halfword signed comparison: \( a < b \) ? 0xffff : 0.

Returns
Returns 0xffff if \( a < b \), else returns 0.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts result is ffff if ‘a’ part < ‘b’ part, and 0000 otherwise. For example __vcmlts2(0x1234aba5, 0x1234aba6) returns 0x0000ffff.

__device__ unsigned int __vcmlts4 (unsigned int a, unsigned int b)
Performs per-byte signed comparison.

Returns
Returns 0xff if \( a < b \), else returns 0.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts result is ff if ‘a’ part < ‘b’ part, and 00 otherwise. For example __vcmlts4(0x1234aba5, 0x1234aba6) returns 0x000000ff.
**__device__ unsigned int __vcmpltu2 (unsigned int a, unsigned int b)**

Performs per-halfword unsigned comparison: \( a < b ? 0xffffff : 0 \).

**Returns**

Returns 0xffffff if \( a < b \), else returns 0.

**Description**

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts result is ffff if ‘a’ part < ‘b’ part, and 0000 otherwise. For example __vcmpltu2(0x1234aba5, 0x1234aba6) returns 0x0000ffff.

**__device__ unsigned int __vcmpltu4 (unsigned int a, unsigned int b)**

Performs per-byte unsigned comparison.

**Returns**

Returns 0xff if \( a < b \), else returns 0.

**Description**

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts result is ff if ‘a’ part < ‘b’ part, and 00 otherwise. For example __vcmpltu4(0x1234aba5, 0x1234aba6) returns 0x000000ff.

**__device__ unsigned int __vcmpne2 (unsigned int a, unsigned int b)**

Performs per-halfword (un)signed comparison: \( a \neq b ? 0xffffff : 0 \).

**Returns**

Returns 0xffffff if \( a \neq b \), else returns 0.

**Description**

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts result is ffff if ‘a’ part \( \neq \) ‘b’ part, and 0000 otherwise. For example __vcmpne2(0x1234aba5, 0x1234aba6) returns 0x0000ffff.
__device__ unsigned int __vcmpne4 (unsigned int a, unsigned int b)
Performs per-byte (un)signed comparison.

Returns
Returns 0xff if a != b, else returns 0.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts result is ff if ‘a’ part != ‘b’ part, and 00 otherwise. For example __vcmlts4(0x1234aba5, 0x1234aba6) returns 0x000000ff.

__device__ unsigned int __vhaddu2 (unsigned int a, unsigned int b)
Performs per-halfword unsigned average computation.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes, then computes unsigned average of corresponding parts. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vhaddu4 (unsigned int a, unsigned int b)
Computes per-byte unsigned average.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. then computes unsigned average of corresponding parts. Partial results are recombined and returned as unsigned int.
__host__ __device__ unsigned int __viaddmax_s16x2 (const unsigned int a, const unsigned int b, const unsigned int c)
Performs per-halfword max(a + b, c).

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. These 2 byte parts are interpreted as signed shorts. For corresponding parts function performs an add and compare: max(a_part + b_part), c_part) Partial results are recombined and returned as unsigned int.

__host__ __device__ unsigned int __viaddmax_s16x2_relu (const unsigned int a, const unsigned int b, const unsigned int c)
Performs per-halfword max(max(a + b, c), 0).

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. These 2 byte parts are interpreted as signed shorts. For corresponding parts function performs an add, followed by a max with relu: max(max(a_part + b_part), c_part), 0) Partial results are recombined and returned as unsigned int.

__host__ __device__ int __viaddmax_s32 (const int a, const int b, const int c)
Computes max(a + b, c).

Returns
Returns computed value.

Description
Calculates the sum of signed integers a and b and takes the max with c.
__host__ __device__ int __viaddmax_s32_relu (const int a, const int b, const int c)
Computes max(max(a + b, c), 0).

Returns
Returns computed value.

Description
Calculates the sum of signed integers a and b and takes the max with c. If the result is less than 0 then is returned.

__host__ __device__ unsigned int __viaddmax_u16x2 (const unsigned int a, const unsigned int b, const unsigned int c)
Performs per-halfword max(a + b, c).

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. These 2 byte parts are interpreted as unsigned shorts. For corresponding parts function performs an add and compare: max(a_part + b_part), c_part] Partial results are recombined and returned as unsigned int.

__host__ __device__ unsigned int __viaddmax_u32 (const unsigned int a, const unsigned int b, const unsigned int c)
Computes max(a + b, c).

Returns
Returns computed value.

Description
Calculates the sum of unsigned integers a and b and takes the max with c.
__host____device__ unsigned int __viaddmin_s16x2 (const unsigned int a, const unsigned int b, const unsigned int c)
Performs per-halfword min(a + b, c).

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. These 2 byte parts are interpreted as signed shorts. For corresponding parts function performs an add and compare: min(a_part + b_part), c_part] Partial results are recombined and returned as unsigned int.

__host____device__ unsigned int __viaddmin_s16x2_relu (const unsigned int a, const unsigned int b, const unsigned int c)
Performs per-halfword max(min(a + b, c), 0).

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. These 2 byte parts are interpreted as signed shorts. For corresponding parts function performs an add, followed by a min with relu: max[min(a_part + b_part), c_part], 0] Partial results are recombined and returned as unsigned int.

__host____device__ int __viaddmin_s32 (const int a, const int b, const int c)
Computes min(a + b, c).

Returns
Returns computed value.

Description
Calculates the sum of signed integers a and b and takes the min with c.
__host__ __device__ int __viaddmin_s32_relu (const int a, const int b, const int c)
Computes max(min(a + b, c), 0).

Returns
Returns computed value.

Description
Calculates the sum of signed integers a and b and takes the min with c. If the result is less than 0 then is returned.

__host__ __device__ unsigned int __viaddmin_u16x2 (const unsigned int a, const unsigned int b, const unsigned int c)
Performs per-halfword min(a + b, c).

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. These 2 byte parts are interpreted as unsigned shorts. For corresponding parts function performs an add and compare: min(a_part + b_part), c_part) Partial results are recombined and returned as unsigned int.

__host__ __device__ unsigned int __viaddmin_u32 (const unsigned int a, const unsigned int b, const unsigned int c)
Computes min(a + b, c).

Returns
Returns computed value.

Description
Calculates the sum of unsigned integers a and b and takes the min with c.
__host__ device__ unsigned int __vibmax_s16x2 (const unsigned int a, const unsigned int b, const bool *pred_hi, const bool *pred_lo)

Performs per-halfword max(a, b), also sets the value pointed to by pred_hi and pred_lo to the per-halfword result of (a >= b).

Returns
Returns computed values.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. These 2 byte parts are interpreted as signed shorts. For corresponding parts function performs a maximum ( = max(a_part, b_part) ). Partial results are recombined and returned as unsigned int. Sets the value pointed to by pred_hi to the value (a_high_part >= b_high_part). Sets the value pointed to by pred_lo to the value (a_low_part >= b_low_part).

__host__ device__ int __vibmax_s32 (const int a, const int b, const bool *pred)

Computes max(a, b), also sets the value pointed to by pred to (a >= b).

Returns
Returns computed values.

Description
Calculates the maximum of a and b of two signed ints. Also sets the value pointed to by pred to the value (a >= b).

__host__ device__ unsigned int __vibmax_u16x2 (const unsigned int a, const unsigned int b, const bool *pred_hi, const bool *pred_lo)

Performs per-halfword max(a, b), also sets the value pointed to by pred_hi and pred_lo to the per-halfword result of (a >= b).

Returns
Returns computed values.
Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. These 2 byte parts are interpreted as unsigned shorts. For corresponding parts function performs a maximum (\( max(a\text{-part}, b\text{-part}) \)). Partial results are recombined and returned as unsigned int. Sets the value pointed to by \texttt{pred\_hi} to the value \([a\text{-high\_part} \geq b\text{-high\_part}]\). Sets the value pointed to by \texttt{pred\_lo} to the value \([a\text{-low\_part} \geq b\text{-low\_part}]\).

\begin{verbatim}
__host____device__ unsigned int __vibmax_u32
(const unsigned int a, const unsigned int b, const bool *pred)
\end{verbatim}

Computes \( max(a, b) \), also sets the value pointed to by \texttt{pred} to \([a \geq b]\).

Returns
Returns computed values.

Description
Calculates the maximum of \(a\) and \(b\) of two unsigned ints. Also sets the value pointed to by \texttt{pred} to the value \([a \geq b]\).

\begin{verbatim}
__host____device__ unsigned int __vibmin_s16x2
(const unsigned int a, const unsigned int b, const bool *pred_hi, const bool *pred_lo)
\end{verbatim}

Performs per-halfword \( min(a, b) \), also sets the value pointed to by \texttt{pred\_hi} and \texttt{pred\_lo} to the per-halfword result of \([a \leq b]\).

Returns
Returns computed values.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. These 2 byte parts are interpreted as signed shorts. For corresponding parts function performs a maximum (\( max(a\text{-part}, b\text{-part}) \)). Partial results are recombined and returned as unsigned int. Sets the value pointed to by \texttt{pred\_hi} to the value \([a\text{-high\_part} \leq b\text{-high\_part}]\). Sets the value pointed to by \texttt{pred\_lo} to the value \([a\text{-low\_part} \leq b\text{-low\_part}]\).
__host____device__ int __vibmin_s32 (const int a, const int b, const bool *pred)
Computes min(a, b), also sets the value pointed to by pred to (a <= b).

Returns
Returns computed values.

Description
Calculates the minimum of a and b of two signed ints. Also sets the value pointed to by pred to the value [a <= b].

__host____device__ unsigned int __vibmin_u16x2 (const unsigned int a, const unsigned int b, const bool *pred_hi, const bool *pred_lo)
Performs per-halfword min(a, b), also sets the value pointed to by pred_hi and pred_lo to the per-halfword result of [a <= b].

Returns
Returns computed values.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. These 2 byte parts are interpreted as unsigned shorts. For corresponding parts function performs a maximum (= max[a_part, b_part]). Partial results are recombined and returned as unsigned int. Sets the value pointed to by pred_hi to the value [a_high_part <= b_high_part]. Sets the value pointed to by pred_lo to the value [a_low_part <= b_low_part].

__host____device__ unsigned int __vibmin_u32 (const unsigned int a, const unsigned int b, const bool *pred)
Computes min(a, b), also sets the value pointed to by pred to (a <= b).

Returns
Returns computed values.
Description
Calculates the minimum of \(a\) and \(b\) of two unsigned ints. Also sets the value pointed to by \(pred\) to the value \((a \leq b)\).

```c
__host__ __device__ unsigned int __vimax3_s16x2
(const unsigned int a, const unsigned int b, const unsigned int c)
```
Performs per-halfword \(\max(\max(a, b), c)\).

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. These 2 byte parts are interpreted as signed shorts. For corresponding parts function performs a 3-way \(\max(\max(a_{\text{part}}, b_{\text{part}}), c_{\text{part}})\). Partial results are recombined and returned as unsigned int.

```c
__host__ __device__ unsigned int __vimax3_s16x2_relu
(const unsigned int a, const unsigned int b, const unsigned int c)
```
Performs per-halfword \(\max(\max(\max(a, b), c), 0)\).

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. These 2 byte parts are interpreted as signed shorts. For corresponding parts function performs a three-way \(\max(a_{\text{part}}, b_{\text{part}}, c_{\text{part}}, 0)\). Partial results are recombined and returned as unsigned int.
\textbf{__host\_\_device\_\_ int \_\_vimax3\_\_s32 (const int a, const int b, const int c)}

Computes \(\max(\max(a, b), c)\).

\textbf{Returns}

Returns computed value.

\textbf{Description}

Calculates the 3-way max of signed integers \(a\), \(b\) and \(c\).

\textbf{__host\_\_device\_\_ int \_\_vimax3\_\_s32\_\_relu (const int a, const int b, const int c)}

Computes \(\max(\max(\max(a, b), c), 0)\).

\textbf{Returns}

Returns computed value.

\textbf{Description}

Calculates the maximum of three signed ints, if this is less than 0 then 0 is returned.

\textbf{__host\_\_device\_\_ unsigned int \_\_vimax3\_\_u16x2 (const unsigned int a, const unsigned int b, const unsigned int c)}

Performs per-halfword \(\max(\max(a, b), c)\).

\textbf{Returns}

Returns computed value.

\textbf{Description}

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. These 2 byte parts are interpreted as unsigned shorts. For corresponding parts function performs a 3-way max \(\max(= \max(\max(a\_part, b\_part), c\_part))\). Partial results are recombined and returned as unsigned int.
__host__ __device__ unsigned int __vimax3_u32 (const unsigned int a, const unsigned int b, const unsigned int c)
Computes max(max(a, b), c).

Returns
Returns computed value.

Description
Calculates the 3-way max of unsigned integers \(a, b\) and \(c\).

__host__ __device__ unsigned int __vimax_s16x2_relu (const unsigned int a, const unsigned int b)
Performs per-halfword max(max(a, b), 0).

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. These 2 byte parts are interpreted as signed shorts. For corresponding parts function performs a max with relu (\(=\) max(a_part, b_part, 0)). Partial results are recombined and returned as unsigned int.

__host__ __device__ int __vimax_s32_relu (const int a, const int b)
Computes max(max(a, b), 0).

Returns
Returns computed value.

Description
Calculates the maximum of \(a\) and \(b\) of two signed ints, if this is less than 0 then 0 is returned.
__host__ __device__ unsigned int __vimin3_s16x2
(const unsigned int a, const unsigned int b, const unsigned int c)
Performs per-halfword min(min(a, b), c).

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. These 2 byte parts are interpreted as signed shorts. For corresponding parts function performs a 3-way min ( = min(min(a_part, b_part), c_part) ). Partial results are recombined and returned as unsigned int.

__host__ __device__ unsigned int __vimin3_s16x2_relu (const unsigned int a, const unsigned int b, const unsigned int c)
Performs per-halfword max(min(min(a, b), c), 0).

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. These 2 byte parts are interpreted as signed shorts. For corresponding parts function performs a three-way min with relu ( = max(min(a_part, b_part, c_part), 0) ). Partial results are recombined and returned as unsigned int.

__host__ __device__ int __vimin3_s32 (const int a, const int b, const int c)
Computes min(min(a, b), c).

Returns
Returns computed value.
Description
Calculates the 3-way min of signed integers a, b and c.

__host____device__ int __vimin3_s32_relu (const int a, const int b, const int c)
Computes max(min(min(a, b), c), 0).

Returns
Returns computed value.

Description
Calculates the minimum of three signed ints, if this is less than 0 then 0 is returned.

__host____device__ unsigned int __vimin3_u16x2 (const unsigned int a, const unsigned int b, const unsigned int c)
Performs per-halfword min(min(a, b), c).

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. These 2 byte parts are interpreted as unsigned shorts. For corresponding parts function performs a 3-way min \( = \min(\min(a_{\text{part}}, b_{\text{part}}), c_{\text{part}}) \). Partial results are recombed and returned as unsigned int.

__host____device__ unsigned int __vimin3_u32 (const unsigned int a, const unsigned int b, const unsigned int c)
Computes min(min(a, b), c).

Returns
Returns computed value.
Description
Calculates the 3-way min of unsigned integers a, b and c.

__host__ __device__ unsigned int __vimin_s16x2_relu (const unsigned int a, const unsigned int b)
Performs per-halfword max(min(a, b), 0).

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. These 2 byte parts are interpreted as signed shorts. For corresponding parts function performs a min with relu ( = max(min[a_part, b_part], 0) ). Partial results are recombined and returned as unsigned int.

__host__ __device__ int __vimin_s32_relu (const int a, const int b)
Computes max(min(a, b), 0).

Returns
Returns computed value.

Description
Calculates the minimum of a and b of two signed ints, if this is less than 0 then 0 is returned.

__device__ unsigned int __vmaxs2 (unsigned int a, unsigned int b)
Performs per-halfword signed maximum computation.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function computes signed maximum. Partial results are recombined and returned as unsigned int.
__device__ unsigned int __vmaxs4 (unsigned int a, unsigned int b)
Computes per-byte signed maximum.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function computes signed maximum. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vmaxu2 (unsigned int a, unsigned int b)
Performs per-halfword unsigned maximum computation.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function computes unsigned maximum. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vmaxu4 (unsigned int a, unsigned int b)
Computes per-byte unsigned maximum.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function computes unsigned maximum. Partial results are recombined and returned as unsigned int.
__device__ unsigned int __vmins2 (unsigned int a, unsigned int b)
Performs per-halfword signed minimum computation.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function computes signed minimum. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vmins4 (unsigned int a, unsigned int b)
Computes per-byte signed minimum.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function computes signed minimum. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vminu2 (unsigned int a, unsigned int b)
Performs per-halfword unsigned minimum computation.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function computes unsigned minimum. Partial results are recombined and returned as unsigned int.
__device__ unsigned int __vminu4 (unsigned int a, unsigned int b)
Computes per-byte unsigned minimum.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function computes unsigned minimum. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vneg2 (unsigned int a)
Computes per-halfword negation.

Returns
Returns computed value.

Description
Splits 4 bytes of argument into 2 parts, each consisting of 2 bytes. For each part function computes negation. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vneg4 (unsigned int a)
Performs per-byte negation.

Returns
Returns computed value.

Description
Splits 4 bytes of argument into 4 parts, each consisting of 1 byte. For each part function computes negation. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vnegss2 (unsigned int a)
Computes per-halfword negation with signed saturation.

Returns
Returns computed value.
Description
Splits 4 bytes of argument into 2 parts, each consisting of 2 bytes. For each part function computes negation. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vnegss4 (unsigned int a)
Performs per-byte negation with signed saturation.

Returns
Returns computed value.

Description
Splits 4 bytes of argument into 4 parts, each consisting of 1 byte. For each part function computes negation. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vsads2 (unsigned int a, unsigned int b)
Performs per-halfword sum of absolute difference of signed.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function computes absolute difference and sum it up. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vsads4 (unsigned int a, unsigned int b)
Computes per-byte sum of abs difference of signed.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function computes absolute difference and sum it up. Partial results are recombined and returned as unsigned int.
__device__ unsigned int __vsadu2 (unsigned int a, unsigned int b)
Computes per-halfword sum of abs diff of unsigned.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function computes absolute differences and returns sum of those differences.

__device__ unsigned int __vsadu4 (unsigned int a, unsigned int b)
Computes per-byte sum of abs difference of unsigned.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function computes absolute differences and returns sum of those differences.

__device__ unsigned int __vseteq2 (unsigned int a, unsigned int b)
Performs per-halfword (un)signed comparison.

Returns
Returns 1 if a = b, else returns 0.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function performs comparison ’a’ part == ’b’ part. If both equalities are satisfied, function returns 1.
__device__ unsigned int __vseteq4 (unsigned int a, unsigned int b)
Performs per-byte (un)signed comparison.

Returns
Returns 1 if a = b, else returns 0.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function performs comparison ‘a’ part == ‘b’ part. If both equalities are satisfied, function returns 1.

__device__ unsigned int __vsetges2 (unsigned int a, unsigned int b)
Performs per-halfword signed comparison.

Returns
Returns 1 if a >= b, else returns 0.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function performs comparison ‘a’ part >= ‘b’ part. If both inequalities are satisfied, function returns 1.

__device__ unsigned int __vsetges4 (unsigned int a, unsigned int b)
Performs per-byte signed comparison.

Returns
Returns 1 if a >= b, else returns 0.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function performs comparison ‘a’ part >= ‘b’ part. If both inequalities are satisfied, function returns 1.
__device__ unsigned int __vsetgeu2 (unsigned int a, unsigned int b)
Performs per-halfword unsigned minimum unsigned comparison.

Returns
Returns 1 if a >= b, else returns 0.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function performs comparison ‘a’ part >= ‘b’ part. If both inequalities are satisfied, function returns 1.

__device__ unsigned int __vsetgeu4 (unsigned int a, unsigned int b)
Performs per-byte unsigned comparison.

Returns
Returns 1 if a >= b, else returns 0.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function performs comparison ‘a’ part >= ‘b’ part. If both inequalities are satisfied, function returns 1.

__device__ unsigned int __vsetgts2 (unsigned int a, unsigned int b)
Performs per-halfword signed comparison.

Returns
Returns 1 if a > b, else returns 0.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function performs comparison ‘a’ part > ‘b’ part. If both inequalities are satisfied, function returns 1.
__device__ unsigned int __vsetgts4 (unsigned int a, unsigned int b)
Performs per-byte signed comparison.

Returns
Returns 1 if a > b, else returns 0.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function performs comparison ‘a’ part > ‘b’ part. If both inequalities are satisfied, function returns 1.

__device__ unsigned int __vsetgtu2 (unsigned int a, unsigned int b)
Performs per-halfword unsigned comparison.

Returns
Returns 1 if a > b, else returns 0.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function performs comparison ‘a’ part > ‘b’ part. If both inequalities are satisfied, function returns 1.

__device__ unsigned int __vsetgtu4 (unsigned int a, unsigned int b)
Performs per-byte unsigned comparison.

Returns
Returns 1 if a > b, else returns 0.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function performs comparison ‘a’ part > ‘b’ part. If both inequalities are satisfied, function returns 1.
__device__ unsigned int __vsetles2 (unsigned int a, unsigned int b)
Performs per-halfword unsigned minimum computation.

Returns
Returns 1 if a <= b, else returns 0.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function performs comparison ‘a’ part <= ‘b’ part. If both inequalities are satisfied, function returns 1.

__device__ unsigned int __vsetles4 (unsigned int a, unsigned int b)
Performs per-byte signed comparison.

Returns
Returns 1 if a <= b, else returns 0.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function performs comparison ‘a’ part <= ‘b’ part. If both inequalities are satisfied, function returns 1.

__device__ unsigned int __vsetleu2 (unsigned int a, unsigned int b)
Performs per-halfword signed comparison.

Returns
Returns 1 if a <= b, else returns 0.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function performs comparison ‘a’ part <= ‘b’ part. If both inequalities are satisfied, function returns 1.
__device__ unsigned int __vsetleu4 (unsigned int a, unsigned int b)
Performs per-byte unsigned comparison.

Returns
Returns 1 if a <= b, else returns 0.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts, function performs comparison 'a' part <= 'b' part. If both inequalities are satisfied, function returns 1.

__device__ unsigned int __vsetlts2 (unsigned int a, unsigned int b)
Performs per-halfword signed comparison.

Returns
Returns 1 if a < b, else returns 0.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts, function performs comparison 'a' part <= 'b' part. If both inequalities are satisfied, function returns 1.

__device__ unsigned int __vsetlts4 (unsigned int a, unsigned int b)
Performs per-byte signed comparison.

Returns
Returns 1 if a < b, else returns 0.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts, function performs comparison 'a' part <= 'b' part. If both inequalities are satisfied, function returns 1.
__device__ unsigned int __vsetltu2 (unsigned int a, unsigned int b)
Performs per-halfword unsigned comparison.

Returns
Returns 1 if a < b, else returns 0.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function performs comparison 'a' part <= 'b' part. If both inequalities are satisfied, function returns 1.

__device__ unsigned int __vsetltu4 (unsigned int a, unsigned int b)
Performs per-byte unsigned comparison.

Returns
Returns 1 if a < b, else returns 0.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function performs comparison 'a' part <= 'b' part. If both inequalities are satisfied, function returns 1.

__device__ unsigned int __vsetne2 (unsigned int a, unsigned int b)
Performs per-halfword (un)signed comparison.

Returns
Returns 1 if a != b, else returns 0.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function performs comparison 'a' part != 'b' part. If both conditions are satisfied, function returns 1.
__device__ unsigned int __vsetne4 (unsigned int a, unsigned int b)
Performs per-byte (un)signed comparison.

Returns
Returns 1 if a != b, else returns 0.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function performs comparison 'a' part != 'b' part. If both conditions are satisfied, function returns 1.

__device__ unsigned int __vsub2 (unsigned int a, unsigned int b)
Performs per-halfword (un)signed subtraction, with wrap-around.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function performs subtraction. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vsub4 (unsigned int a, unsigned int b)
Performs per-byte subtraction.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function performs subtraction. Partial results are recombined and returned as unsigned int.
__device__ unsigned int __vsubss2 (unsigned int a, unsigned int b)
Performs per-halfword (un)signed subtraction, with signed saturation.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function performs subtraction with signed saturation. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vsubss4 (unsigned int a, unsigned int b)
Performs per-byte subtraction with signed saturation.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function performs subtraction with signed saturation. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vsubus2 (unsigned int a, unsigned int b)
Performs per-halfword subtraction with unsigned saturation.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function performs subtraction with unsigned saturation. Partial results are recombined and returned as unsigned int.
__device__ unsigned int __vsubus4 (unsigned int a, unsigned int b)
Performs per-byte subtraction with unsigned saturation.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function performs subtraction with unsigned saturation. Partial results are recombined and returned as unsigned int.
Chapter 2. Data Structures

Here are the data structures with brief descriptions:

__half
   __half data type
__half2
   __half2 data type
__half2_raw
   __half2_raw data type
__half_raw
   __half_raw data type

_nv_bfloat16
   Nv_bfloat16 datatype
_nv_bfloat162
   Nv_bfloat162 datatype
_nv_bfloat162_raw
   __nv_bfloat162_raw data type
_nv_bfloat16_raw
   __nv_bfloat16_raw data type

_nv_fp8_e4m3
   __nv_fp8_e4m3 datatype
_nv_fp8_e5m2
   __nv_fp8_e5m2 datatype
_nv_fp8x2_e4m3
   __nv_fp8x2_e4m3 datatype
_nv_fp8x2_e5m2
   __nv_fp8x2_e5m2 datatype
_nv_fp8x4_e4m3
   __nv_fp8x4_e4m3 datatype
_nv_fp8x4_e5m2
   __nv_fp8x4_e5m2 datatype
2.1. __half Struct Reference

__half data type

This structure implements the datatype for storing half-precision floating-point numbers. The structure implements assignment, arithmetic and comparison operators, and type conversions. 16 bits are being used in total: 1 sign bit, 5 bits for the exponent, and the significand is being stored in 10 bits. The total precision is 11 bits. There are 15361 representable numbers within the interval [0.0, 1.0], endpoints included. On average we have log10(2**11) ~ 3.311 decimal digits.

The objective here is to provide IEEE754-compliant implementation of binary16 type and arithmetic with limitations due to device HW not supporting floating-point exceptions.

unsigned short __half::__x

Protected storage variable contains the bits of floating-point data.

__half
__half __host__ __device__ ( const unsigned long long  val ) [inline]
Construct __half from unsigned long long input using default round-to-nearest-even rounding mode.

__half
__half __host__ __device__ ( const long long  val ) [inline]
Construct __half from long long input using default round-to-nearest-even rounding mode.

__half
__half __host__ __device__ ( const unsignedlong  val ) [inline]
Construct __half from unsigned long input using default round-to-nearest-even rounding mode.

__half
__half __host__ __device__ ( const long  val ) [inline]
Construct `__half` from long input using default round-to-nearest-even rounding mode.

`__half` __host__ __device__ (const unsigned int val) [inline]

Construct `__half` from unsigned int input using default round-to-nearest-even rounding mode.

`__half` __host__ __device__ (const int val) [inline]

Construct `__half` from int input using default round-to-nearest-even rounding mode.

`__half` __host__ __device__ (const unsigned short val) [inline]

Construct `__half` from unsigned short integer input using default round-to-nearest-even rounding mode.

`__half` __host__ __device__ (const short val) [inline]

Construct `__half` from short integer input using default round-to-nearest-even rounding mode.

`__half` __host__ __device__ (const double f) [inline]

Construct `__half` from double input using default round-to-nearest-even rounding mode.

`__half` __host__ __device__ (const float f) [inline]

Construct `__half` from float input using default round-to-nearest-even rounding mode.
__half
__half ( const __half_raw& hr ) [inline]
Constructor from __half_raw.

__half
__half __host__ __device__ [ ] [inline]
Constructor by default.

__host____device__operator __half_raw ()
Description
Type cast to __half_raw operator with volatile input.

__host____device__operator __half_raw ()
Description
Type cast to __half_raw operator.

__host____device__constexpr operator bool ()
Description
Conversion operator to bool data type. +0 and -0 inputs convert to false. Non-zero inputs convert to true.

__host____device__operator char ()
Description
Conversion operator to an implementation defined char data type. Using round-toward-zero rounding mode.
Detects signedness of the char type and proceeds accordingly, see further details in signed and unsigned char operators.
__host__ device__operator float ()

Description
Type cast to float operator.

__host__ device__operator int ()

Description
Conversion operator to int data type. Using round-toward-zero rounding mode.
See __half2int_rz__(half) for further details

__host__ device__operator long ()

Description
Conversion operator to long data type. Using round-toward-zero rounding mode.

__host__ device__operator long long ()

Description
Conversion operator to long long data type. Using round-toward-zero rounding mode.
See __half2ll_rz__(half) for further details

__host__ device__operator short ()

Description
Conversion operator to short data type. Using round-toward-zero rounding mode.
See __half2short_rz__(half) for further details

__host__ device__operator signed char ()

Description
Conversion operator to signed char data type. Using round-toward-zero rounding mode.
See __half2char_rz__(half) for further details
__host____device__operator unsigned char ()

Description
Conversion operator to unsigned char data type. Using round-toward-zero rounding mode.
See __half2uchar_rz(__half) for further details

__host____device__operator unsigned int ()

Description
Conversion operator to unsigned int data type. Using round-toward-zero rounding mode.
See __half2uint_rz(__half) for further details

__host____device__operator unsigned long ()

Description
Conversion operator to unsigned long data type. Using round-toward-zero rounding mode.

__host____device__operator unsigned long long ()

Description
Conversion operator to unsigned long long data type. Using round-toward-zero rounding mode.
See __half2ull_rz(__half) for further details

__host____device__operator unsigned short ()

Description
Conversion operator to unsigned short data type. Using round-toward-zero rounding mode.
See __half2ushort_rz(__half) for further details
__host__ __device__ operator= (const unsigned long long val)

Description
Type cast from unsigned long long assignment operator, using default round-to-nearest-even rounding mode.

__host__ __device__ operator= (const long long val)

Description
Type cast from long long assignment operator, using default round-to-nearest-even rounding mode.

__host__ __device__ operator= (const unsigned int val)

Description
Type cast from unsigned int assignment operator, using default round-to-nearest-even rounding mode.

__host__ __device__ operator= (const int val)

Description
Type cast from int assignment operator, using default round-to-nearest-even rounding mode.

__host__ __device__ operator= (const unsigned short val)

Description
Type cast from unsigned short assignment operator, using default round-to-nearest-even rounding mode.
__host__ __device__ operator= (const short val)

Description
Type cast from short assignment operator, using default round-to-nearest-even rounding mode.

__host__ __device__ operator= (const double f)

Description
Type cast to __half assignment operator from double input using default round-to-nearest-even rounding mode.

__host__ __device__ operator= (const float f)

Description
Type cast to __half assignment operator from float input using default round-to-nearest-even rounding mode.

__host__ __device__ operator= (const __half_raw hr)

Description
Assignment operator from volatile __half_raw to volatile __half.

__host__ __device__ operator= (const __half_raw hr)

Description
Assignment operator from __half_raw to volatile __half.

__host__ __device__ operator= (const __half_raw hr)

Description
Assignment operator from __half_raw.
2.2. __half2 Struct Reference

__half2 data type

This structure implements the datatype for storing two half-precision floating-point numbers. The structure implements assignment, arithmetic and comparison operators, and type conversions.

- NOTE: __half2 is visible to non-nvcc host compilers

```
struct __half__half2::x
Storage field holding lower __half part.

struct __half__half2::y
Storage field holding upper __half part.

__half2
__half2 __host__ __device__ ( const __half2_raw& h2r ) [inline]
Constructor from __half2_raw

__half2
__half2 __host__ __device__ ( const __half2& src ) [inline]
Copy constructor

__half2
__half2 __host__ __device__ ( const __half& a, const __half& b ) [inline]
Constructor from two __half variables

__half2
__half2 __host__ __device__ () [inline]
Constructor by default.
```
__host____device__operator __half2_raw ()

Description
Conversion operator to __half2_raw

__host____device__operator= (const __half2_raw h2r)

Description
Assignment operator from __half2_raw

__host____device__operator= (const __half2 src)

Description
Copy assignment operator

2.3. __half2_raw Struct Reference
__half2_raw data type
Type allows static initialization of half2 until it becomes a builtin type.

- Note: this initialization is as a bit-field representation of half2, and not a conversion from short2 to half2. Such representation will be deprecated in a future version of CUDA.
- Note: this is visible to non-nvcc compilers, including C-only compilations

2.4. __half_raw Struct Reference
__half_raw data type
Type allows static initialization of half until it becomes a builtin type.

- Note: this initialization is as a bit-field representation of half, and not a conversion from short to half. Such representation will be deprecated in a future version of CUDA.
- Note: this is visible to non-nvcc compilers, including C-only compilations
2.5. __nv_bfloat16 Struct Reference

_nv_bfloat16 datatype

This structure implements the datatype for storing nv_bfloat16 floating-point numbers. The structure implements assignment operators and type conversions. 16 bits are being used in total: 1 sign bit, 8 bits for the exponent, and the significand is being stored in 7 bits. The total precision is 8 bits.

unsigned short __nv_bfloat16::__x

Protected storage variable contains the bits of floating-point data.

__nv_bfloat16

__nv_bfloat16 __host__ __device__ [ unsigned long long ] [inline]

Construct __nv_bfloat16 from unsigned long long input using default round-to-nearest-even rounding mode.

__nv_bfloat16

__nv_bfloat16 __host__ __device__ [ long long ] [inline]

Construct __nv_bfloat16 from long long input using default round-to-nearest-even rounding mode.

__nv_bfloat16

__nv_bfloat16 __host__ __device__ [ long long ] [inline]

Construct __nv_bfloat16 from unsigned long input using default round-to-nearest-even rounding mode.

__nv_bfloat16

__nv_bfloat16 __host__ __device__ [ const long ] [inline]

Construct __nv_bfloat16 from long input using default round-to-nearest-even rounding mode.
__nv_bfloat16
__nv_bfloat16 __host__ __device__ [ unsigned int val ] [inline]
Construct __nv_bfloat16 from unsigned int input using default round-to-nearest-even rounding mode.

__nv_bfloat16
__nv_bfloat16 __host__ __device__ [ int val ] [inline]
Construct __nv_bfloat16 from int input using default round-to-nearest-even rounding mode.

__nv_bfloat16
__nv_bfloat16 __host__ __device__ [ unsigned short val ] [inline]
Construct __nv_bfloat16 from unsigned short integer input using default round-to-nearest-even rounding mode.

__nv_bfloat16
__nv_bfloat16 __host__ __device__ [ short val ] [inline]
Construct __nv_bfloat16 from short integer input using default round-to-nearest-even rounding mode.

__nv_bfloat16
__nv_bfloat16 __host__ __device__ [ const double f ] [inline]
Construct __nv_bfloat16 from double input using default round-to-nearest-even rounding mode.

__nv_bfloat16
__nv_bfloat16 __host__ __device__ [ const float f ] [inline]
Construct __nv_bfloat16 from float input using default round-to-nearest-even rounding mode.
__nv_bfloat16
__nv_bfloat16 __host__ __device__ ( const __nv_bfloat16_raw& hr ) [inline]
Constructor from __nv_bfloat16_raw.

__nv_bfloat16
__nv_bfloat16 __host__ __device__ () [inline]
Constructor by default.

__host____device__operator __nv_bfloat16_raw ()
Description
Type cast to __nv_bfloat16_raw operator with volatile input.

__host____device__operator __nv_bfloat16_raw ()
Description
Type cast to __nv_bfloat16_raw operator.

__host____device__ constexpr operator bool ()
Description
Conversion operator to bool data type. +0 and -0 inputs convert to false. Non-zero inputs convert to true.

__host____device__operator char ()
Description
Conversion operator to an implementation defined char data type. Using round-toward-zero rounding mode.
Detects signedness of the char type and proceeds accordingly, see further details in signed and unsigned char operators.
__host__ __device__ operator float ()

Description
Type cast to float operator.

__host__ __device__ operator int ()

Description
Conversion operator to int data type. Using round-toward-zero rounding mode.
See __bfloat162int_rz( __nv_bfloat16) for further details

__host__ __device__ operator long ()

Description
Conversion operator to long data type. Using round-toward-zero rounding mode.

__host__ __device__ operator long long ()

Description
Conversion operator to long long data type. Using round-toward-zero rounding mode.
See __bfloat162ll_rz( __nv_bfloat16) for further details

__host__ __device__ operator short ()

Description
Conversion operator to short data type. Using round-toward-zero rounding mode.
See __bfloat162short_rz( __nv_bfloat16) for further details

__host__ __device__ operator signed char ()

Description
Conversion operator to signed char data type. Using round-toward-zero rounding mode.
See __bfloat162char_rz( __nv_bfloat16) for further details
__host__ __device__ operator unsigned char ()

Description
Conversion operator to unsigned char data type. Using round-toward-zero rounding mode.
See __bfloat162uchar_rz(__nv_bfloat16) for further details

__host__ __device__ operator unsigned int ()

Description
Conversion operator to unsigned int data type. Using round-toward-zero rounding mode.
See __bfloat162uint_rz(__nv_bfloat16) for further details

__host__ __device__ operator unsigned long ()

Description
Conversion operator to unsigned long data type. Using round-toward-zero rounding mode.

__host__ __device__ operator unsigned long long ()

Description
Conversion operator to unsigned long long data type. Using round-toward-zero rounding mode.
See __bfloat162ull_rz(__nv_bfloat16) for further details

__host__ __device__ operator unsigned short ()

Description
Conversion operator to unsigned short data type. Using round-toward-zero rounding mode.
See __bfloat162ushort_rz(__nv_bfloat16) for further details
__host__ __device__ operator= (unsigned long long val)

Description
Type cast from unsigned long long assignment operator, using default round-to-nearest-even rounding mode.

__host__ __device__ operator= (long long val)

Description
Type cast from long long assignment operator, using default round-to-nearest-even rounding mode.

__host__ __device__ operator= (unsigned int val)

Description
Type cast from unsigned int assignment operator, using default round-to-nearest-even rounding mode.

__host__ __device__ operator= (int val)

Description
Type cast from int assignment operator, using default round-to-nearest-even rounding mode.

__host__ __device__ operator= (unsigned short val)

Description
Type cast from unsigned short assignment operator, using default round-to-nearest-even rounding mode.

__host__ __device__ operator= (short val)

Description
Type cast from short assignment operator, using default round-to-nearest-even rounding mode.
__host__ __device__ operator= (const double f)

Description
Type cast to __nv_bfloat16 assignment operator from double input using default round-to-nearest-even rounding mode.

__host__ __device__ operator= (const float f)

Description
Type cast to __nv_bfloat16 assignment operator from float input using default round-to-nearest-even rounding mode.

__host__ __device__ operator= (const __nv_bfloat16_raw hr)

Description
Assignment operator from volatile __nv_bfloat16_raw to volatile __nv_bfloat16 .

__host__ __device__ operator= (const __nv_bfloat16_raw hr)

Description
Assignment operator from __nv_bfloat16_raw to volatile __nv_bfloat16 .

__host__ __device__ operator= (const __nv_bfloat16_raw hr)

Description
Assignment operator from __nv_bfloat16_raw .
2.6. __nv_bfloat162 Struct Reference

nv_bfloat162 datatype

This structure implements the datatype for storing two nv_bfloat16 floating-point numbers. The structure implements assignment, arithmetic and comparison operators, and type conversions.

- NOTE: __nv_bfloat162 is visible to non-nvcc host compilers

```c
struct __nv_bfloat16 __nv_bfloat162::x
```
Storage field holding lower __nv_bfloat16 part.

```c
struct __nv_bfloat16 __nv_bfloat162::y
```
Storage field holding upper __nv_bfloat16 part.

```c
__nv_bfloat162
__nv_bfloat162 __host__ __device__ ( const __nv_bfloat162_raw& h2r )
```
Constructor from __nv_bfloat162_raw

```c
__nv_bfloat162
__nv_bfloat162 __host__ __device__ ( const __nv_bfloat16& src )
```
Copy constructor

```c
__nv_bfloat162
__nv_bfloat162 __host__ __device__ ( const __nv_bfloat16& a, const __nv_bfloat16& b ) [inline]
```
Constructor from two __nv_bfloat16 variables

```c
__nv_bfloat162
__nv_bfloat162 __host__ __device__ Constructor by default.
```
__host__ __device__ operator __nv_bfloat162_raw ()

Description
Conversion operator to __nv_bfloat162_raw

__host__ __device__ operator= (const __nv_bfloat162_raw h2r)

Description
Assignment operator from __nv_bfloat162_raw

__host__ __device__ operator= (const __nv_bfloat162 src)

Description
Copy assignment operator

2.7. __nv_bfloat162_raw Struct Reference

__nv_bfloat162_raw data type
Type allows static initialization of nv_bfloat162 until it becomes a builtin type.

‣ Note: this initialization is as a bit-field representation of nv_bfloat162, and not a conversion from short2 to nv_bfloat162. Such representation will be deprecated in a future version of CUDA.

‣ Note: this is visible to non-nvcc compilers, including C-only compilations

2.8. __nv_bfloat16_raw Struct Reference

__nv_bfloat16_raw data type
Type allows static initialization of nv_bfloat16 until it becomes a builtin type.

‣ Note: this initialization is as a bit-field representation of nv_bfloat16, and not a conversion from short to nv_bfloat16. Such representation will be deprecated in a future version of CUDA.

‣ Note: this is visible to non-nvcc compilers, including C-only compilations
2.9. __nv_fp8_e4m3 Struct Reference

__nv_fp8_e4m3 datatype

This structure implements the datatype for storing fp8 floating-point numbers of e4m3 kind: with 1 sign, 4 exponent, 1 implicit and 3 explicit mantissa bits. The encoding doesn’t support Infinity. NaNs are limited to 0x7F and 0xFF values.

The structure implements converting constructors and operators.

__nv_fp8_storage_t __nv_fp8_e4m3::__x

Storage variable contains the fp8 floating-point data.

__nv_fp8_e4m3

__nv_fp8_e4m3 __host__ __device__  [ const long long int val ] [inline, explicit]

Constructor from long long int data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8_e4m3

__nv_fp8_e4m3 __host__ __device__  [ const long int  val ] [inline, explicit]

Constructor from long int data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8_e4m3

__nv_fp8_e4m3 __host__ __device__  [ const int  val ] [inline, explicit]

Constructor from int data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8_e4m3

__nv_fp8_e4m3 __host__ __device__  [ const short int val ] [inline, explicit]
Constructor from short int data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8_e4m3
__nv_fp8_e4m3 __host__ __device__ ( const unsigned long long int val ) [inline, explicit]
Constructor from unsigned long long int data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8_e4m3
__nv_fp8_e4m3 __host__ __device__ ( const unsigned long int val ) [inline, explicit]
Constructor from unsigned long int data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8_e4m3
__nv_fp8_e4m3 __host__ __device__ ( const unsigned int val ) [inline, explicit]
Constructor from unsigned int data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8_e4m3
__nv_fp8_e4m3 __host__ __device__ ( const unsigned short int val ) [inline, explicit]
Constructor from unsigned short int data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8_e4m3
__nv_fp8_e4m3 __host__ __device__ ( const double f ) [inline, explicit]
Constructor from double data type, relies on __NV_SATFINITE behavior for out-of-range values.
__nv_fp8_e4m3
__nv_fp8_e4m3 __host__ __device__ [ const float f ] [inline, explicit]
Constructor from float data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8_e4m3
__nv_fp8_e4m3 __host__ __device__ [ const __nv_bfloat16 f ] [inline, explicit]
Constructor from __nv_bfloat16 data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8_e4m3
__nv_fp8_e4m3 __host__ __device__ [ const __half f ] [inline, explicit]
Constructor from __half data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8_e4m3
__nv_fp8_e4m3()
Constructor by default.

__host____device__operator __half ()
Description
Conversion operator to __half data type.

__host____device__operator __nv_bfloat16 ()
Description
Conversion operator to __nv_bfloat16 data type.
__host____device__operator bool ()

Description
Conversion operator to bool data type. +0 and -0 inputs convert to false. Non-zero inputs convert to true.

__host____device__operator char ()

Description
Conversion operator to an implementation defined char data type.
Detects signedness of the char type and proceeds accordingly, see further details in signed and unsigned char operators.
Clamps inputs to the output range. NaN inputs convert to zero.

__host____device__operator double ()

Description
Conversion operator to double data type.

__host____device__operator float ()

Description
Conversion operator to float data type.

__host____device__operator int ()

Description
Conversion operator to int data type. NaN inputs convert to zero.

__host____device__operator long int ()

Description
Conversion operator to long int data type. Clamps too large inputs to the output range. NaN inputs convert to zero if output type is 32-bit. NaN inputs convert to 0x8000000000000000ULL if output type is 64-bit.
__host__ __device__ operator long long int ()

Description
Conversion operator to long long int data type. NaN inputs convert to 0x8000000000000000ULL.

__host__ __device__ operator short int ()

Description
Conversion operator to short int data type. NaN inputs convert to zero.

__host__ __device__ operator signed char ()

Description
Conversion operator to signed char data type. Clamps too large inputs to the output range. NaN inputs convert to zero.

__host__ __device__ operator unsigned char ()

Description
Conversion operator to unsigned char data type. Clamps negative and too large inputs to the output range. NaN inputs convert to zero.

__host__ __device__ operator unsigned int ()

Description
Conversion operator to unsigned int data type. Clamps negative inputs to zero. NaN inputs convert to zero.

__host__ __device__ operator unsigned long int ()

Description
Conversion operator to unsigned long int data type. Clamps negative and too large inputs to the output range. NaN inputs convert to zero if output type is 32-bit. NaN inputs convert to 0x8000000000000000ULL if output type is 64-bit.
__host__ __device__ operator unsigned long long int ()

**Description**

Conversion operator to unsigned long long int data type. Clamps negative inputs to zero. NaN inputs convert to 0x8000000000000000ULL.

__host__ __device__ operator unsigned short int ()

**Description**

Conversion operator to unsigned short int data type. Clamps negative inputs to zero. NaN inputs convert to zero.

### 2.10. __nv_fp8_e5m2 Struct Reference

__nv_fp8_e5m2 datatype

This structure implements the datatype for handling fp8 floating-point numbers of e5m2 kind: with 1 sign, 5 exponent, 1 implicit and 2 explicit mantissa bits.

The structure implements converting constructors and operators.

__nv_fp8_storage_t __nv_fp8_e5m2::__x

Storage variable contains the fp8 floating-point data.

__nv_fp8_e5m2 __nv_fp8_e5m2 __host__ __device__ ( const long long int val ) [inline, explicit]

Constructor from long long int data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8_e5m2 __nv_fp8_e5m2 __host__ __device__ ( const long int val ) [inline, explicit]

Constructor from long int data type, relies on __NV_SATFINITE behavior for out-of-range values.
__nv_fp8_e5m2
__nv_fp8_e5m2 __host__ __device__  [ const int  val ] [inline, explicit]
Constructor from int data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8_e5m2
__nv_fp8_e5m2 __host__ __device__  [ const short int val ] [inline, explicit]
Constructor from short int data type.

__nv_fp8_e5m2
__nv_fp8_e5m2 __host__ __device__  [ const unsigned long long int val ] [inline, explicit]
Constructor from unsigned long long int data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8_e5m2
__nv_fp8_e5m2 __host__ __device__  [ const unsigned long int val ] [inline, explicit]
Constructor from unsigned long int data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8_e5m2
__nv_fp8_e5m2 __host__ __device__  [ const unsigned int val ] [inline, explicit]
Constructor from unsigned int data type, relies on __NV_SATFINITE behavior for out-of-range values.
__nv_fp8_e5m2
__nv_fp8_e5m2  __host__   __device__  [ const unsigned short int val ] [inline, explicit]
Constructor from unsigned short int data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8_e5m2
__nv_fp8_e5m2  __host__   __device__  [ const double f ] [inline, explicit]
Constructor from double data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8_e5m2
__nv_fp8_e5m2  __host__   __device__  [ const float f ] [inline, explicit]
Constructor from float data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8_e5m2
__nv_fp8_e5m2  __host__   __device__  [ const __nv_bfloat16 f ] [inline, explicit]
Constructor from __nv_bfloat16 data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8_e5m2
__nv_fp8_e5m2  __host__   __device__  [ const __half f ] [inline, explicit]
Constructor from __half data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8_e5m2
__nv_fp8_e5m2()
Constructor by default.
__host__ device__operator __half ()

Description
Conversion operator to __half data type.

__host__ device__operator __nv_bfloat16 ()

Description
Conversion operator to __nv_bfloat16 data type.

__host__ device__operator bool ()

Description
Conversion operator to bool data type. +0 and -0 inputs convert to false. Non-zero inputs convert to true.

__host__ device__operator char ()

Description
Conversion operator to an implementation defined char data type. Detects signedness of the char type and proceeds accordingly, see further details in signed and unsigned char operators.

Clamps inputs to the output range. NaN inputs convert to zero.

__host__ device__operator double ()

Description
Conversion operator to double data type.

__host__ device__operator float ()

Description
Conversion operator to float data type.
__host___device__operator int ()

Description
Conversion operator to int data type. Clamps too large inputs to the output range. NaN inputs convert to zero.

__host___device__operator long int ()

Description
Conversion operator to long int data type. Clamps too large inputs to the output range. NaN inputs convert to zero if output type is 32-bit. NaN inputs convert to 0x8000000000000000ULL if output type is 64-bit.

__host___device__operator long long int ()

Description
Conversion operator to long long int data type. Clamps too large inputs to the output range. NaN inputs convert to 0x8000000000000000ULL.

__host___device__operator short int ()

Description
Conversion operator to short int data type. Clamps too large inputs to the output range. NaN inputs convert to zero.

__host___device__operator signed char ()

Description
Conversion operator to signed char data type. Clamps too large inputs to the output range. NaN inputs convert to zero.

__host___device__operator unsigned char ()

Description
Conversion operator to unsigned char data type. Clamps negative and too large inputs to the output range. NaN inputs convert to zero.
__host____device__operator unsigned int ()

Description
Conversion operator to `unsigned int` data type. Clamps negative and too large inputs to the output range. NaN inputs convert to zero.

__host____device__operator unsigned long int ()

Description
Conversion operator to `unsigned long int` data type. Clamps negative and too large inputs to the output range. NaN inputs convert to zero if output type is 32-bit. NaN inputs convert to 0x8000000000000000ULL if output type is 64-bit.

__host____device__operator unsigned long long int ()

Description
Conversion operator to `unsigned long long int` data type. Clamps negative and too large inputs to the output range. NaN inputs convert to 0x8000000000000000ULL.

__host____device__operator unsigned short int ()

Description
Conversion operator to `unsigned short int` data type. Clamps negative and too large inputs to the output range. NaN inputs convert to zero.

2.11. __nv_fp8x2_e4m3 Struct Reference

__nv_fp8x2_e4m3 datatype
This structure implements the datatype for storage and operations on the vector of two fp8 values of e4m3 kind each: with 1 sign, 4 exponent, 1 implicit and 3 explicit mantissa bits. The encoding doesn’t support Infinity. NaNs are limited to 0x7F and 0xFF values.

__nv_fp8x2_storage_t __nv_fp8x2_e4m3::__x
Storage variable contains the vector of two fp8 floating-point data values.
__nv_fp8x2_e4m3
__nv_fp8x2_e4m3 __host__ __device__  [ const double2 f ] [inline, explicit]
Constructor from double2 data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8x2_e4m3
__nv_fp8x2_e4m3 __host__ __device__  [ const float2 f ] [inline, explicit]
Constructor from float2 data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8x2_e4m3
__nv_fp8x2_e4m3 __host__ __device__  [ const __nv_bfloat162 f ] [inline, explicit]
Constructor from __nv_bfloat162 data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8x2_e4m3
__nv_fp8x2_e4m3 __host__ __device__  [ const __half2 f ] [inline, explicit]
Constructor from __half2 data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8x2_e4m3
__nv_fp8x2_e4m3 [ ]
Constructor by default.

__host____device__operator__half2 ()

Description
Conversion operator to __half2 data type.
__host__ __device__ operator float2 ()

Description
Conversion operator to float2 data type.

2.12. __nv_fp8x2_e5m2 Struct Reference
__nv_fp8x2_e5m2 datatype
This structure implements the datatype for handling two fp8 floating-point numbers of e5m2 kind each: with 1 sign, 5 exponent, 1 implicit and 2 explicit mantissa bits.
The structure implements converting constructors and operators.

__nv_fp8x2_storage_t __nv_fp8x2_e5m2::__x
Storage variable contains the vector of two fp8 floating-point data values.

__nv_fp8x2_e5m2
__nv_fp8x2_e5m2 __host__ __device__ ( const double2 f ) [inline, explicit]
Constructor from double2 data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8x2_e5m2
__nv_fp8x2_e5m2 __host__ __device__ ( const float2 f ) [inline, explicit]
Constructor from float2 data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8x2_e5m2
__nv_fp8x2_e5m2 __host__ __device__ ( const __nv_bfloat162 f ) [inline, explicit]
Constructor from __nv_bfloat162 data type, relies on __NV_SATFINITE behavior for out-of-range values.
__nv_fp8x2_e5m2
__nv_fp8x2_e5m2 __host__ __device__ (const __half2 f) [inline, explicit]
Constructor from __half2 data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8x2_e5m2
__nv_fp8x2_e5m2 ()
Constructor by default.

__host__ __device__ operator __half2 ()
Description
Conversion operator to __half2 data type.

__host__ __device__ operator float2 ()
Description
Conversion operator to float2 data type.

2.13. __nv_fp8x4_e4m3 Struct Reference
__nv_fp8x4_e4m3 datatype
This structure implements the datatype for storage and operations on the vector of four fp8 values of e4m3 kind each: with 1 sign, 4 exponent, 1 implicit and 3 explicit mantissa bits. The encoding doesn’t support Infinity. NaNs are limited to 0x7F and 0xFFF values.

__nv_fp8x4_storage_t __nv_fp8x4_e4m3::__x
Storage variable contains the vector of four fp8 floating-point data values.

__nv_fp8x4_e4m3
__nv_fp8x4_e4m3 __host__ __device__ (const double4 f) [inline, explicit]
Constructor from `double4` vector data type, relies on `__NV_SATFINITE` behavior for out-of-range values.

```cpp
__nv_fp8x4_e4m3
__nv_fp8x4_e4m3 __host__ __device__ ( const float4 f ) [inline, explicit]
```
Constructor from `float4` vector data type, relies on `__NV_SATFINITE` behavior for out-of-range values.

```cpp
__nv_fp8x4_e4m3
__nv_fp8x4_e4m3 __host__ __device__ ( const float f ) [inline, explicit]
```
Constructor from a pair of `__nv_bfloat16` data type values, relies on `__NV_SATFINITE` behavior for out-of-range values.

```cpp
__nv_fp8x4_e4m3
__nv_fp8x4_e4m3 __host__ __device__ ( const __nv_bfloat16 flo, const __nv_bfloat16 fhi ) [inline, explicit]
```
Constructor from a pair of `__half` data type values, relies on `__NV_SATFINITE` behavior for out-of-range values.

```cpp
__nv_fp8x4_e4m3
__nv_fp8x4_e4m3 __host__ __device__ ( const __half flo, const __half fhi ) [inline, explicit]
```
Constructor by default.

```cpp
__host__ __device__ operator float4 ()
```

**Description**

Conversion operator to `float4` vector data type.

### 2.14. __nv_fp8x4_e5m2 Struct Reference

__nv_fp8x4_e5m2 datatype

This structure implements the datatype for handling four `fp8` floating-point numbers of `e5m2` kind each: with 1 sign, 5 exponent, 1 implicit and 2 explicit mantissa bits.
The structure implements converting constructors and operators.

__nv_fp8x4_storage_t __nv_fp8x4_e5m2::__x

Storage variable contains the vector of four fp8 floating-point data values.

__nv_fp8x4_e5m2
__nv_fp8x4_e5m2 __host__ __device__ ( const double4 f ) [inline, explicit]

Constructor from double4 vector data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8x4_e5m2
__nv_fp8x4_e5m2 __host__ __device__ ( const float4 f ) [inline, explicit]

Constructor from float4 vector data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8x4_e5m2
__nv_fp8x4_e5m2 __host__ __device__ ( const __nv_bfloat162 flo, const __nv_bfloat162 fhi ) [inline, explicit]

Constructor from a pair of __nv_bfloat162 data type values, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8x4_e5m2
__nv_fp8x4_e5m2 __host__ __device__ ( const __half2 flo, const __half2 fhi ) [inline, explicit]

Constructor from a pair of __half2 data type values, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8x4_e5m2
__nv_fp8x4_e5m2 ()

Constructor by default.
__host__ __device__ operator float4 ()

Description
Conversion operator to float4 vector data type.
Chapter 3. Data Fields

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

```plaintext

__half()
  __half

__half2()
  __half2

__nv_bfloat16()
  __nv_bfloat16

__nv_bfloat162()
  __nv_bfloat162

__nv_fp8_e4m3()
  __nv_fp8_e4m3

__nv_fp8_e5m2()
  __nv_fp8_e5m2

__nv_fp8x2_e4m3()
  __nv_fp8x2_e4m3

__nv_fp8x2_e5m2()
  __nv_fp8x2_e5m2

__nv_fp8x4_e4m3()
  __nv_fp8x4_e4m3

__nv_fp8x4_e5m2()
  __nv_fp8x4_e5m2

__x
  __nv_fp8x2_e4m3
  __nv_fp8x2_e5m2
  __nv_bfloat16
  __nv_fp8_e4m3
  __nv_fp8_e5m2
```

CUDA Math API v12.4 | 475
0

operator __half()
  _nv_fp8_e5m2
  _nv_fp8_e4m3

operator __half2()
  _nv_fp8x2_e4m3
  _nv_fp8x2_e5m2

operator __half2_raw()
  __half2

operator __half_raw()
  __half

operator __nv_bfloat16()
  _nv_fp8_e5m2
  _nv_fp8_e4m3

operator __nv_bfloat162_raw()
  __nv_bfloat162

operator __nv_bfloat16_raw()
  __nv_bfloat16

operator bool()
  __nv_bfloat16
  _nv_fp8_e5m2
  _nv_fp8_e4m3
  __half

operator char()
  __nv_bfloat16
  _nv_fp8_e5m2
  _nv_fp8_e4m3
  __half
  __nv_bfloat16

operator double()
  __nv_bfloat16
  _nv_fp8_e5m2
  _nv_fp8_e4m3

operator float()
  __nv_bfloat16
  _nv_fp8_e5m2
  _nv_fp8_e4m3
  __half
  __nv_bfloat16

operator float2()
  _nv_fp8x2_e5m2
  _nv_fp8x2_e4m3

operator float4()
  _nv_fp8x4_e5m2
  _nv_fp8x4_e4m3
operator int()
    __nv_fp8_e4m3
    __half
    __nv_bfloat16
    __nv_fp8_e5m2

operator long()
    __half
    __nv_bfloat16

operator long int()
    __nv_fp8_e5m2
    __nv_fp8_e4m3

operator long long()
    __half
    __nv_bfloat16

operator long long int()
    __nv_fp8_e5m2
    __nv_fp8_e4m3

operator short()
    __half
    __nv_bfloat16

operator short int()
    __nv_fp8_e5m2
    __nv_fp8_e4m3

operator signed char()
    __nv_fp8_e5m2
    __nv_fp8_e4m3
    __half
    __nv_bfloat16

operator unsigned char()
    __nv_fp8_e5m2
    __nv_fp8_e4m3
    __half
    __nv_bfloat16

operator unsigned int()
    __nv_fp8_e5m2
    __nv_fp8_e4m3
    __half
    __nv_bfloat16

operator unsigned long()
    __half
    __nv_bfloat16

operator unsigned long int()
    __nv_fp8_e5m2
__nv_fp8_e4m3
operator unsigned long long()
__nv_bfloat16
half
operator unsigned long long int()
__nv_fp8_e5m2
__nv_fp8_e4m3
operator unsigned short()
__nv_bfloat16
half
operator unsigned short int()
__nv_fp8_e4m3
__nv_fp8_e5m2
operator=()
half
__nv_bfloat16
half
__nv_bfloat16
half
__nv_bfloat16
half
__nv_bfloat16
half
__nv_bfloat16
half2
__nv_bfloat16
__nv_bfloat162
half
__nv_bfloat162
half
__nv_bfloat162
half
__nv_bfloat16
X

x

__half2
__nv_bfloat162

Y

y

__half2
__nv_bfloat162
Notice
This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation (“NVIDIA”) makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material [defined below], code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice. Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer (“Terms of Sale”). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

OpenCL
OpenCL is a trademark of Apple Inc. used under license to the Khronos Group Inc.

Trademarks
NVIDIA and the NVIDIA logo are trademarks or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright
© 2007-2024 NVIDIA Corporation & affiliates. All rights reserved.