CUDA Math API

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 fp8 data type of e4m3 kind ............................................................................................ 2

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 .................................................................................................................................................. 3

__nv_fp8x2_storage_t ............................................................................................................................................... 3

__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 ................................................................................................................................. 10

__nv_fp8_e5m2::__nv_fp8_e5m2 ................................................................................................................................. 10
1.1.3. C++ struct for handling vector type of two fp8 values of e5m2 kind................................. 13
__nv_fp8x2_e5m2::__x............................................................................................................ 13

1.1.4. C++ struct for handling vector type of four fp8 values of e5m2 kind............................. 14
__nv_fp8x4_e5m2::__x............................................................................................................ 15

1.1.5. C++ struct for handling fp8 data type of e4m3 kind.................................................... 16
__nv_fp8_e4m3::__x................................................................................................................16
1.2. Half Precision Intrinsics

1.2.1. Half Arithmetic Functions

1.2.2. Half2 Arithmetic Functions

atomicAdd
1.2.3. Half Comparison Functions.................................................................36
   __heq.........................................................................................................37
   __hequ......................................................................................................37
   __hge.........................................................................................................37
   __hgeu......................................................................................................38
   __hgt.........................................................................................................38
   __hgtu......................................................................................................39
   __hisinf....................................................................................................39
   __hisnan.................................................................................................40
   __hle.........................................................................................................40
   __hleu......................................................................................................40
   __hlt.........................................................................................................41
   __hltu......................................................................................................41
   __hmax.......................................................................................................42
   __hmax_nan.............................................................................................42
   __hmin.......................................................................................................42
   __hmin_nan.............................................................................................42
   __hne.........................................................................................................42
   __hneu......................................................................................................43
   __hneu......................................................................................................43
1.2.4. Half2 Comparison Functions.............................................................43
   __hbeq2....................................................................................................44
   __hbequ2...............................................................................................44
   __hbge2....................................................................................................45
   __hbgeu2...............................................................................................45
   __hbgt2....................................................................................................46
   __hbgtu2...............................................................................................46
   __hble2....................................................................................................47
   __hbleu2...............................................................................................47
1.2.5. Half Precision Conversion and Data Movement

- __double2half................................................................. 64
- __float2half................................................................. 65
- __float2half2_rn............................................................ 64
__highs2half2.......................................................................................................................... 82
__int2half_rd............................................................................................................................82
__int2half_rn............................................................................................................................83
__int2half_ru............................................................................................................................83
__int2half_rz............................................................................................................................83
__ldca.......................................................................................................................................84
__ldca.......................................................................................................................................84
__ldcg.......................................................................................................................................84
__ldcg.......................................................................................................................................84
__ldcs.......................................................................................................................................85
__ldcs.......................................................................................................................................85
__ldcs.......................................................................................................................................85
__ldcv.......................................................................................................................................85
__ldcv.......................................................................................................................................85
__ldg.........................................................................................................................................86
__ldg.........................................................................................................................................86
__ldg.........................................................................................................................................86
__ldlu.......................................................................................................................................86
__ldlu.......................................................................................................................................86
__ll2half_rd..............................................................................................................................87
__ll2half_rn..............................................................................................................................87
__ll2half_ru..............................................................................................................................88
__ll2half_rz..............................................................................................................................88
__low2float...............................................................................................................................89
__low2half................................................................................................................................89
__low2half2...............................................................................................................................89
__lowhigh2highlow..................................................................................................................90
__lows2half2.............................................................................................................................90
__shfl_down_sync...................................................................................................................91
__shfl_down_sync...................................................................................................................92
__shfl_sync...............................................................................................................................93
__shfl_sync...............................................................................................................................93
__shfl_up_sync........................................................................................................................94
__shfl_up_sync........................................................................................................................95
__shfl_xor_sync.......................................................................................................................96
__shfl_xor_sync.......................................................................................................................97
__short2half_rd.......................................................................................................................97
__short2half_rn.......................................................................................................................98
__short2half_ru.......................................................................................................................98
__short2half_rz.......................................................................................................................99
__short_as_half.......................................................................................................................99
__stcg.....................................................................................................................................100
__stcg.....................................................................................................................................100
__stcs.....................................................................................................................................100
__stcs.....................................................................................................................................100
__stwb....................................................................................................................................101
__stwb....................................................................................................................................101
__stwt.....................................................................................................................................101
__stwt.....................................................................................................................................101
__uint2half_rd.......................................................................................................................102
__uint2half_rn.......................................................................................................................102
__uint2half_ru.......................................................................................................................102
__uint2half_rz........................................................................................................................103
__ull2half_rd..........................................................................................................................103
__ull2half_rn..........................................................................................................................104
__ull2half_ru..........................................................................................................................104
__ull2half_rz..........................................................................................................................105
__ushort2half_rd...................................................................................................................105
__ushort2half_rn...................................................................................................................106
__ushort2half_ru...................................................................................................................106
__ushort2half_rz...................................................................................................................107
__ushort_as_half...................................................................................................................107

1.2.6. Half Math Functions.......................................................................................................107
hceil........................................................................................................................................108
hcos........................................................................................................................................108
hexp........................................................................................................................................108
hexp10....................................................................................................................................109
hexp2......................................................................................................................................109
hfloor......................................................................................................................................109
hlog........................................................................................................................................ 110
hlog10.....................................................................................................................................110
hlog2...................................................................................................................................... 111
hrcp........................................................................................................................................ 111
hrint........................................................................................................................................111
hrsqrt..................................................................................................................................... 112
hsin.........................................................................................................................................112
hsqrt.......................................................................................................................................113
htrunc.....................................................................................................................................113
1.2.7. Half2 Math Functions ..................................................................................................... 113
    h2ceil...................................................................................................................................... 114
    h2cos...................................................................................................................................... 114
    h2exp...................................................................................................................................... 114
    h2exp10.................................................................................................................................. 115
    h2exp2.................................................................................................................................... 115
    h2floor.................................................................................................................................... 116
    h2log...................................................................................................................................... 116
    h2log10................................................................................................................................... 116
    h2log2..................................................................................................................................... 117
    h2rcp...................................................................................................................................... 117
    h2rint...................................................................................................................................... 118
    h2rsqrt................................................................................................................................... 118
    h2sin....................................................................................................................................... 118
    h2sqrt..................................................................................................................................... 119
    h2trunc................................................................................................................................... 119

1.3. Bfloat16 Precision Intrinsics ........................................................................................... 119
    Bfloat16 Arithmetic Functions................................................................................................. 120
    Bfloat162 Arithmetic Functions ............................................................................................ 120
    Bfloat16 Comparison Functions.............................................................................................. 120
    Bfloat162 Comparison Functions ........................................................................................... 120
    Bfloat16 Precision Conversion and Data Movement .............................................................. 120
    Bfloat16 Math Functions.......................................................................................................... 120
    Bfloat162 Math Functions........................................................................................................ 120
1.3.1. Bfloat16 Arithmetic Functions ....................................................................................... 120
    __h2div................................................................................................................................... 120
    __habs.................................................................................................................................... 120
    __hadd.................................................................................................................................... 121
    __hadd_rn.............................................................................................................................. 121
    __hadd_sat............................................................................................................................ 121
    __hddiv.................................................................................................................................. 122
    __hfma................................................................................................................................... 122
    __hfma_relu........................................................................................................................... 122
    __hfma_sat............................................................................................................................ 123
    __hmul................................................................................................................................... 123
    __hmul_rn.............................................................................................................................. 123
    __hmul_sat............................................................................................................................ 124
    __hneg.................................................................................................................................... 124
1.3.2. Bfloat16 Arithmetic Functions

- __habs2.................................................................................................................................. 126
- __hadd2.................................................................................................................................. 126
- __hadd2_rn............................................................................................................................ 127
- __hadd2_sat........................................................................................................................... 127
- __hcmadd............................................................................................................................. 127
- __hfma2.................................................................................................................................. 128
- __hfma2_relu......................................................................................................................... 128
- __hfma2_sat.......................................................................................................................... 129
- __hmul2.................................................................................................................................. 130
- __hmul2_rn............................................................................................................................ 130
- __hmul2_sat.......................................................................................................................... 130
- __hneg2.................................................................................................................................. 131
- __hsub2.................................................................................................................................. 131
- __hsub2_rn............................................................................................................................ 131
- __hsub2_sat........................................................................................................................... 131

1.3.3. Bfloat16 Comparison Functions

- __heq...................................................................................................................................... 133
- __hequ.................................................................................................................................... 133
- __hge...................................................................................................................................... 134
- __hgeu..................................................................................................................................... 134
- __hgt...................................................................................................................................... 135
- __hgtu.................................................................................................................................... 135
- __hisinf................................................................................................................................... 136
- __hisnan.................................................................................................................................. 136
- __hle....................................................................................................................................... 136
- __hleu..................................................................................................................................... 137
- __hlt....................................................................................................................................... 137
- __hltu...................................................................................................................................... 138
- __hmax..................................................................................................................................... 138
- __hmax_nan........................................................................................................................... 139
- __hmin..................................................................................................................................... 139
- __hmin_nan............................................................................................................................ 139
1.3.4. Bfloat162 Comparison Functions

__hbeq2.......................................................................................................................... 140
__hbequ2.......................................................................................................................... 141
__hbge2............................................................................................................................ 142
__hbgeu2.......................................................................................................................... 142
__hbg2t2............................................................................................................................ 143
__hbgtu2............................................................................................................................. 144
__hble2............................................................................................................................... 144
__hbleu2............................................................................................................................. 145
__hblt2................................................................................................................................. 145
__hbltu2............................................................................................................................... 146
__hnne2............................................................................................................................... 147
__hnneu2............................................................................................................................. 147
__heq2................................................................................................................................. 148
__heq2_mask......................................................................................................................... 148
__hequ2............................................................................................................................... 149
__hequ2_mask....................................................................................................................... 149
__hge2................................................................................................................................. 150
__hge2_mask......................................................................................................................... 150
__hgeu2............................................................................................................................... 151
__hgeu2_mask....................................................................................................................... 151
__hgt2................................................................................................................................. 152
__hgt2_mask......................................................................................................................... 152
__hgtu2............................................................................................................................... 153
__hgtu2_mask....................................................................................................................... 153
__hisnan2........................................................................................................................... 153
__hle2................................................................................................................................. 154
__hle2_mask......................................................................................................................... 155
__hleu2............................................................................................................................... 155
__hleu2_mask....................................................................................................................... 156
__hlt2................................................................................................................................. 156
__hlt2_mask......................................................................................................................... 156
__hltu2............................................................................................................................... 157
__hltu2_mask....................................................................................................................... 157
__hmax2............................................................................................................................... 158
__hmax2_nan......................................................................................................................... 159
1.3.5. Bfloat16 Precision Conversion and Data Movement

__bfloat162float2
__bfloat162bfloat162
__bfloat162float
__bfloat162int_rd
__bfloat162int_rn
__bfloat162int_ru
__bfloat162int_rz
__bfloat162ll_rd
__bfloat162ll_rn
__bfloat162ll_ru
__bfloat162ll_rz
__bfloat162short_rd
__bfloat162short_rn
__bfloat162short_ru
__bfloat162short_rz
__bfloat162uint_rd
__bfloat162uint_rn
__bfloat162uint_ru
__bfloat162uint_rz
__bfloat162ull_rd
__bfloat162ull_rn
__bfloat162ull_ru
__bfloat162ull_rz
__bfloat162ushort_rd
__bfloat162ushort_rn
__bfloat162ushort_ru
__bfloat162ushort_rz
__bfloat16_as_short
__bfloat16_as_ushort
__double2bfloat16
__float2bfloat162_rn
__shfl_down_sync................................................................................................................. 193
__shfl_sync............................................................................................................................ 194
__shfl_sync............................................................................................................................ 195
__shfl_up_sync...................................................................................................................... 196
__shfl_up_sync...................................................................................................................... 197
__shfl_xor_sync.....................................................................................................................198
__shfl_xor_sync.....................................................................................................................199
__short2bfloat16_rd.............................................................................................................. 200
__short2bfloat16_rn.............................................................................................................. 200
__short2bfloat16_ru.............................................................................................................. 201
__short2bfloat16_rz.................................................................................................................201
__short_as_bfloat16..............................................................................................................202
__stcg.....................................................................................................................................202
__stcg.....................................................................................................................................202
__stcs.....................................................................................................................................203
__stcs.....................................................................................................................................203
__stwb....................................................................................................................................203
__stwb....................................................................................................................................203
__stwt.....................................................................................................................................204
__stwt.....................................................................................................................................204
__uint2bfloat16_rd................................................................................................................ 204
__uint2bfloat16_rn................................................................................................................ 205
__uint2bfloat16_ru................................................................................................................ 205
__uint2bfloat16_rz.................................................................................................................206
__ull2bfloat16_rd...................................................................................................................206
__ull2bfloat16_rn...................................................................................................................207
__ull2bfloat16_ru...................................................................................................................207
__ull2bfloat16_rz...................................................................................................................208
__ushort2bfloat16_rd............................................................................................................ 208
__ushort2bfloat16_rn............................................................................................................209
__ushort2bfloat16_ru............................................................................................................209
__ushort2bfloat16_rz............................................................................................................ 210
__ushort_as_bfloat16............................................................................................................210

1.3.6. Bfloat16 Math Functions...........................................................................................210
hceil........................................................................................................................................211
hcos........................................................................................................................................211
hexp........................................................................................................................................211
hexp10.................................................................................................................................212
hexp2 ........................................................................................................................................ 212
hfloor ...................................................................................................................................... 213
hlog ........................................................................................................................................ 213
hlog10 ..................................................................................................................................... 213
hlog2 ...................................................................................................................................... 214
hrcp ........................................................................................................................................ 214
hrint ........................................................................................................................................ 214
hrsqrt ..................................................................................................................................... 215
hsin ......................................................................................................................................... 215
hsqrt ....................................................................................................................................... 216
htrunc ..................................................................................................................................... 216
1.3.7. Bfloat16 Math Functions .............................................................................................. 216
h2ceil ...................................................................................................................................... 217
h2cos ...................................................................................................................................... 217
h2exp ...................................................................................................................................... 217
h2exp10 .................................................................................................................................. 218
h2exp2 .................................................................................................................................... 218
h2floor .................................................................................................................................... 219
h2log ...................................................................................................................................... 219
h2log10 ................................................................................................................................... 219
h2log2 ..................................................................................................................................... 220
h2rcp ...................................................................................................................................... 220
h2rint ...................................................................................................................................... 220
h2rsqrt ................................................................................................................................... 221
h2sin ....................................................................................................................................... 221
h2sqrt ..................................................................................................................................... 222
h2trunc ................................................................................................................................... 222
1.4. Mathematical Functions ..................................................................................................... 223
1.5. Single Precision Mathematical Functions ........................................................................... 223
acosf ........................................................................................................................................ 223
acoshf ...................................................................................................................................... 224
asin .......................................................................................................................................... 224
asinhf ...................................................................................................................................... 225
atan2f ..................................................................................................................................... 225
atanf ...................................................................................................................................... 226
atanhf ..................................................................................................................................... 226
cbrtf ....................................................................................................................................... 227
ceilf ........................................................................................................................................ 227
log1pf ......................................................................................................................................... 245
log10f ......................................................................................................................................... 244
llroundf ...................................................................................................................................... 244
llrintf .......................................................................................................................................... 243
lgammaf .................................................................................................................................... 243
ldexpf ......................................................................................................................................... 242
jnf ............................................................................................................................................... 242
j0f ............................................................................................................................................... 241
j1f ............................................................................................................................................... 241
jnrf .............................................................................................................................................. 242
ldexf ........................................... 242
lgammaf ........................................ 243
lrintf ........................................ 243
llroundf ..................................... 244
log10f ......................................... 244
log1pf ........................................ 245
log2f........................................................................................................................................... 245
logbf........................................................................................................................................... 246
logf............................................................................................................................................. 246
lrintf........................................................................................................................................... 247
lroundf....................................................................................................................................... 247
max............................................................................................................................................ 247
min............................................................................................................................................. 248
modff........................................................................................................................................... 248
nanf............................................................................................................................................ 248
nearbyintf...................................................................................................................................249
nextafterf................................................................................................................................... 249
norm3df..................................................................................................................................... 250
norm4df..................................................................................................................................... 250
normcdf.................................................................................................................................... 251
normcdfinvf................................................................................................................................251
normf......................................................................................................................................... 252
powf........................................................................................................................................... 252
rcbrtf.......................................................................................................................................... 253
remainderf.................................................................................................................................254
remquof..................................................................................................................................... 254
rhypotf........................................................................................................................................255
rintf............................................................................................................................................ 255
rnorm3df....................................................................................................................................256
rnorm4df....................................................................................................................................256
rnormf........................................................................................................................................ 257
roundf........................................................................................................................................ 257
rsqrtf.......................................................................................................................................... 258
scalblnf...................................................................................................................................... 258
scalbnf....................................................................................................................................... 258
signbit........................................................................................................................................ 259
sincosf........................................................................................................................................259
sincospif.....................................................................................................................................260
sinf............................................................................................................................................. 260
sinhf........................................................................................................................................... 261
sinpif.......................................................................................................................................... 261
sqrtnf.......................................................................................................................................... 262
tanf............................................................................................................................................. 262
tanhf........................................................................................................................................... 263
1.6. Double Precision Mathematical Functions

tgammaf .................................................................................................................................... 263
truncf ......................................................................................................................................... 264
y0f ...............................................................................................................................................264
y1f ...............................................................................................................................................264
ynf .............................................................................................................................................. 265

acos............................................................................................................................................266
acosh..........................................................................................................................................266
asin............................................................................................................................................ 267
asinh.......................................................................................................................................... 267
atan............................................................................................................................................ 268
atan2.......................................................................................................................................... 268
atanh..........................................................................................................................................269
cbrt.............................................................................................................................................269
ceil ..............................................................................................................................................270
copysign.....................................................................................................................................270
cos..............................................................................................................................................270
cosh............................................................................................................................................271
cospi..........................................................................................................................................271
cyl_bessel_i0............................................................................................................................. 272
cyl_bessel_i1............................................................................................................................. 272
erf ...............................................................................................................................................272
erfc .............................................................................................................................................273
erfcinv ....................................................................................................................................... 273
erfcx .......................................................................................................................................... 274
erfinv ......................................................................................................................................... 274
exp ..............................................................................................................................................275
exp10 ..........................................................................................................................................275
exp2 .............................................................................................................................................276
expm1 ..........................................................................................................................................276
fabs ............................................................................................................................................. 277
fdim ............................................................................................................................................277
floor ..........................................................................................................................................278
fma ............................................................................................................................................. 278
fmax .......................................................................................................................................... 279
fmin .......................................................................................................................................... 279
fmod .......................................................................................................................................... 280
frexp ......................................................................................................................................... 280
hypot ........................................................................................................................................ 281
ilogb .................................................................................................................................... 281
isfinite ................................................................................................................................. 282
isinf ..................................................................................................................................... 282
isnan ..................................................................................................................................... 283
j0 ......................................................................................................................................... 283
j1 ......................................................................................................................................... 283
jn ......................................................................................................................................... 283
ldexp ................................................................................................................................... 284
lgamma ................................................................................................................................. 285
llrint .................................................................................................................................... 285
llround ................................................................................................................................. 286
log ....................................................................................................................................... 286
log10 ................................................................................................................................. 286
log1p ................................................................................................................................. 287
log2 ..................................................................................................................................... 287
logb ..................................................................................................................................... 288
lrint ..................................................................................................................................... 288
lround ................................................................................................................................. 288
max ..................................................................................................................................... 289
max ..................................................................................................................................... 289
max ..................................................................................................................................... 289
min ..................................................................................................................................... 289
min ..................................................................................................................................... 289
min ..................................................................................................................................... 289
modf ................................................................................................................................... 290
nan ...................................................................................................................................... 291
nearbyint ............................................................................................................................ 291
nextafter .............................................................................................................................. 292
norm ................................................................................................................................. 292
norm3d .............................................................................................................................. 293
norm4d .............................................................................................................................. 293
normcdf ............................................................................................................................. 294
normcdfinv ......................................................................................................................... 294
pow .................................................................................................................................... 295
rcbrt .................................................................................................................................... 296
remainder ............................................................................................................................ 296
remquo ................................................................................................................................. 297
1.7. Integer Mathematical Functions

abs............................. 308
labs............................... 308
llabs.............................. 308
llmax............................. 309
llmin............................. 309
max................................ 309
max................................ 309
max................................ 310
max................................ 310
max................................ 310
max................................ 310
max................................ 310
max................................ 311
max................................ 311
1.8. Single Precision Intrinsics

- \texttt{__cosf}.........................................................................................................................315
- \texttt{__exp10f}.........................................................................................................................316
- \texttt{__expf}.............................................................................................................................316
- \texttt{__fadd_rd}.........................................................................................................................316
- \texttt{__fadd_rn}.........................................................................................................................317
- \texttt{__fadd_ru}.........................................................................................................................317
- \texttt{__fadd_rz}.........................................................................................................................318
- \texttt{__fdiv_rd}.........................................................................................................................318
- \texttt{__fdiv_rn}.........................................................................................................................318
- \texttt{__fdiv_ru}.........................................................................................................................319
- \texttt{__fdiv_rz}.........................................................................................................................319
- \texttt{__fdivdef}........................................................................................................................319
- \texttt{__fmaf_ieee_rd}................................................................................................................320
- \texttt{__fmaf_ieee_rn}................................................................................................................320
- \texttt{__fmaf_ieee_ru}................................................................................................................320
- \texttt{__fmaf_ieee_rz}................................................................................................................321
- \texttt{__fmaf_rd}..........................................................................................................................321
- \texttt{__fmaf_rn}..........................................................................................................................321

\texttt{max}........................................................................................................................................311
\texttt{max}........................................................................................................................................311
\texttt{max}........................................................................................................................................311
\texttt{min}........................................................................................................................................312
\texttt{min}........................................................................................................................................312
\texttt{min}........................................................................................................................................312
\texttt{min}........................................................................................................................................312
\texttt{min}........................................................................................................................................312
\texttt{min}........................................................................................................................................313
\texttt{min}........................................................................................................................................313
\texttt{min}........................................................................................................................................313
\texttt{min}........................................................................................................................................313
\texttt{min}........................................................................................................................................314
\texttt{min}........................................................................................................................................314
\texttt{min}........................................................................................................................................314
\texttt{ullmax}...................................................................................................................................314
\texttt{ullmin}....................................................................................................................................314
\texttt{umax}.....................................................................................................................................315
\texttt{umin}.....................................................................................................................................315
CUDA Math API

1.9. Double Precision Intrinsics

__fadd_rn........................................................................................................ 328
__fadd_ru........................................................................................................ 328
__fadd_rz........................................................................................................ 328
__fdiv_rn......................................................................................................... 329
__fdiv_ru......................................................................................................... 329
__fdiv_rz......................................................................................................... 329
__fmul_rn......................................................................................................... 329
__fmul_ru......................................................................................................... 329
__fmul_rz......................................................................................................... 329
__frsqrt ......................................................................................................... 330
__fsqrt ......................................................................................................... 330
__frsqrt ......................................................................................................... 330
__fsqrt ......................................................................................................... 330
__fsqrt ......................................................................................................... 330
__fsqrt ......................................................................................................... 330
__fsub_rn........................................................................................................ 331
__fsub_ru........................................................................................................ 331
__fsub_rz........................................................................................................ 331
__log10f......................................................................................................... 332
__log2f......................................................................................................... 332
__logf......................................................................................................... 332
__logf......................................................................................................... 332
__logf......................................................................................................... 332
__powf......................................................................................................... 332
__powf......................................................................................................... 332
__powf......................................................................................................... 332
__saturatef.................................................................................................. 333
__sincosf.................................................................................................... 333
__sincosf.................................................................................................... 333
__sinf......................................................................................................... 333
__sinf......................................................................................................... 333
__sinf......................................................................................................... 333
__tanhf........................................................................................................ 334
__tanhf........................................................................................................ 334
__tanhf........................................................................................................ 334

1.9. Double Precision Intrinsics

__daddd_rd................................................................. 333
__daddd_rn................................................................. 334
__daddd_ru................................................................. 334
__daddd_rz................................................................. 334
__dddiv_rn................................................................. 335
__dddiv_rz................................................................. 336
__dmul_rn................................................................. 336
__dmul_rn................................................................. 336

1.9. Double Precision Intrinsics

__fadd_rn........................................................................................................ 328
__fadd_ru........................................................................................................ 328
__fadd_rz........................................................................................................ 328
__fdiv_rn......................................................................................................... 329
__fdiv_ru......................................................................................................... 329
__fdiv_rz......................................................................................................... 329
__fmul_rn......................................................................................................... 329
__fmul_ru......................................................................................................... 329
__fmul_rz......................................................................................................... 329
__frsqrt ......................................................................................................... 330
__fsqrt ......................................................................................................... 330
__frsqrt ......................................................................................................... 330
__fsqrt ......................................................................................................... 330
__frsqrt ......................................................................................................... 330
__fsub_rn........................................................................................................ 331
__fsub_ru........................................................................................................ 331
__fsub_rz........................................................................................................ 331
__log10f......................................................................................................... 332
__log2f......................................................................................................... 332
__logf......................................................................................................... 332
__logf......................................................................................................... 332
__powf......................................................................................................... 332
__powf......................................................................................................... 332
__saturatef.................................................................................................. 333
__sincosf.................................................................................................... 333
__sincosf.................................................................................................... 333
__sinf......................................................................................................... 333
__sinf......................................................................................................... 333
__tanhf........................................................................................................ 334
__tanhf........................................................................................................ 334
__tanhf........................................................................................................ 334

1.9. Double Precision Intrinsics

__daddd_road.................................................................................................. 333
__daddd_rn................................................................. 334
__daddd_ru................................................................. 334
__daddd_rz................................................................. 334
__dddiv_rn................................................................. 335
__dddiv_rz................................................................. 336
__dmul_rn................................................................. 336
__dmul_rn................................................................. 336
1.10. Integer Intrinsics

__brev........................................................................................................................................ 346
__brevll...................................................................................................................................... 346
__byte_perm..............................................................................................................................346
__clz........................................................................................................................................... 347
__clzll.........................................................................................................................................347
__ffs........................................................................................................................................... 347
__ffsll......................................................................................................................................... 348
__funnelshift_l...........................................................................................................................348
__funnelshift_ll..........................................................................................................................348
__hadd....................................................................................................................................... 349
__mul24..................................................................................................................................... 350
__mul64hi.................................................................................................................................. 350
__mulhi......................................................................................................................................350
__popc........................................................................................................................................351
__popcll..................................................................................................................................... 351
__rhadd......................................................................................................................................351
__sad..........................................................................................................................................351
1.11. Type Casting Intrinsics

- __double2float_rd................................................................. 354
- __double2float_rn................................................................. 354
- __double2float_ru................................................................. 354
- __double2float_rz................................................................. 355
- __double2hiint................................................................. 355
- __double2int_rd................................................................. 355
- __double2int_rn................................................................. 356
- __double2int_ru................................................................. 356
- __double2int_rz................................................................. 356
- __double2ll_rd................................................................. 356
- __double2ll_rn................................................................. 357
- __double2ll_ru................................................................. 357
- __double2loint..................................................................... 358
- __double2uint_rd............................................................... 358
- __double2uint_rn............................................................... 358
- __double2uint_ru............................................................... 358
- __double2uint_rz............................................................... 359
- __double2ull_rd............................................................... 359
- __double2ull_rn............................................................... 359
- __double2ull_ru............................................................... 360
- __double2ull_rz............................................................... 360
- __double_as_longlong......................................................... 360
- __float2int_rd............................................................... 361
- __float2int_rn............................................................... 361
- __float2int_ru............................................................... 361
- __float2int_rz............................................................... 361
- __float2ull_rd............................................................... 362
- __float2ull_rn............................................................... 362
- __float2ull_ru............................................................... 362
- __float2ull_rz............................................................... 363
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>__float2uint_rd</td>
<td>363</td>
</tr>
<tr>
<td>__float2uint_rn</td>
<td>363</td>
</tr>
<tr>
<td>__float2uint_ru</td>
<td>363</td>
</tr>
<tr>
<td>__float2uint_rz</td>
<td>364</td>
</tr>
<tr>
<td>__float2ull_rd</td>
<td>364</td>
</tr>
<tr>
<td>__float2ull_rn</td>
<td>364</td>
</tr>
<tr>
<td>__float2ull_ru</td>
<td>365</td>
</tr>
<tr>
<td>__float2ull_rz</td>
<td>365</td>
</tr>
<tr>
<td>__float_as_int</td>
<td>365</td>
</tr>
<tr>
<td>__float_as_uint</td>
<td>365</td>
</tr>
<tr>
<td>__hiloint2double</td>
<td>366</td>
</tr>
<tr>
<td>__int2double_rn</td>
<td>366</td>
</tr>
<tr>
<td>__int2float_rd</td>
<td>366</td>
</tr>
<tr>
<td>__int2float_rn</td>
<td>366</td>
</tr>
<tr>
<td>__int2float_ru</td>
<td>367</td>
</tr>
<tr>
<td>__int2float_rz</td>
<td>367</td>
</tr>
<tr>
<td>__int_as_float</td>
<td>367</td>
</tr>
<tr>
<td>__ll2double_rd</td>
<td>368</td>
</tr>
<tr>
<td>__ll2double_rn</td>
<td>368</td>
</tr>
<tr>
<td>__ll2double_ru</td>
<td>368</td>
</tr>
<tr>
<td>__ll2double_rz</td>
<td>369</td>
</tr>
<tr>
<td>__ll2float_rd</td>
<td>369</td>
</tr>
<tr>
<td>__ll2float_rn</td>
<td>369</td>
</tr>
<tr>
<td>__ll2float_ru</td>
<td>369</td>
</tr>
<tr>
<td>__ll2float_rz</td>
<td>370</td>
</tr>
<tr>
<td>__longlong_as_double</td>
<td>370</td>
</tr>
<tr>
<td>__uint2double_rn</td>
<td>370</td>
</tr>
<tr>
<td>__uint2float_rd</td>
<td>371</td>
</tr>
<tr>
<td>__uint2float_rn</td>
<td>371</td>
</tr>
<tr>
<td>__uint2float_ru</td>
<td>371</td>
</tr>
<tr>
<td>__uint2float_rz</td>
<td>371</td>
</tr>
<tr>
<td>__uint_as_float</td>
<td>372</td>
</tr>
<tr>
<td>__ull2double_rd</td>
<td>372</td>
</tr>
<tr>
<td>__ull2double_rn</td>
<td>372</td>
</tr>
<tr>
<td>__ull2double_ru</td>
<td>373</td>
</tr>
<tr>
<td>__ull2double_rz</td>
<td>373</td>
</tr>
<tr>
<td>__ull2float_rd</td>
<td>373</td>
</tr>
<tr>
<td>__ull2float_rn</td>
<td>374</td>
</tr>
</tbody>
</table>
1.12. SIMD Intrinsics

__vabs2 ................................................................. 375
__vabs4 ................................................................. 375
__vabsdiffs2 ......................................................... 375
__vabsdiffs4 ......................................................... 376
__vabssu2 .............................................................. 376
__vabssu4 .............................................................. 376
__vadd2 ................................................................. 377
__vadd4 ................................................................. 377
__vaddss2 .............................................................. 377
__vaddus2 .............................................................. 377
__vavgs2 ............................................................... 379
__vavgs4 ............................................................... 379
__vavgu2 ............................................................... 379
__vavgu4 ............................................................... 379
__vcmpseq2 ........................................................... 381
__vcmpseq4 ........................................................... 381
__vcmpges2 ........................................................... 381
__vcmpges4 ........................................................... 382
__vcmpgeu2 ........................................................... 382
__vcmpgeu4 ........................................................... 382
__vcmpgtu2 ........................................................... 383
__vcmpgtu4 ........................................................... 383
__vcmplu2 ............................................................. 384
__vcmplsu4 ........................................................... 384
__vcmplu4 ............................................................. 385
__vcmplts2 ........................................................... 385
__vcmplts4 ........................................................... 386
__vcmples2 ........................................................... 386
__vcmples4 ........................................................... 386
__vcmpltu4................................................................................................................................ 386
__vcmpne2.................................................................................................................................. 387
__vcmpne4.................................................................................................................................. 387
__vhaddu2.................................................................................................................................. 387
__vhaddu4.................................................................................................................................. 388
__viaddmax_s16x2.................................................................................................................... 388
__viaddmax_s16x2_relu............................................................................................................ 388
__viaddmax_s32........................................................................................................................ 389
__viaddmax_s32_relu................................................................................................................ 389
__viaddmax_u16x2.................................................................................................................... 389
__viaddmax_u32........................................................................................................................ 390
__viaddmin_s16x2..................................................................................................................... 390
__viaddmin_s16x2_relu............................................................................................................ 390
__viaddmin_s32......................................................................................................................... 391
__viaddmin_s32_relu................................................................................................................ 391
__viaddmin_u16x2..................................................................................................................... 391
__viaddmin_u32......................................................................................................................... 392
__vibmax_s16x2........................................................................................................................ 392
__vibmax_s32............................................................................................................................ 392
__vibmax_u16x2........................................................................................................................ 393
__vibmax_u32............................................................................................................................ 393
__vibmin_s16x2........................................................................................................................ 394
__vibmin_s32............................................................................................................................. 394
__vibmin_u16x2......................................................................................................................... 394
__vibmin_u32............................................................................................................................. 395
__vimax3_s16x2......................................................................................................................... 395
__vimax3_s16x2_relu................................................................................................................ 396
__vimax3_s32............................................................................................................................ 396
__vimax3_s32_relu.................................................................................................................... 396
__vimax3_u16x2........................................................................................................................ 397
__vimax3_u32............................................................................................................................ 397
__vimax_s16x2.......................................................................................................................... 397
__vimax_s32............................................................................................................................. 398
__vimax_s32_relu..................................................................................................................... 398
__vimax3_s16x2_relu................................................................................................................ 399
__vimax3_s32_relu.................................................................................................................... 399
__vimax3_u16x2........................................................................................................................ 399
__vimax_u16x2.......................................................................................................................... 399
__vimin3_u32.............................................................................................................................400
__vimin_s16x2_relu.................................................................................................................. 400
__vimin_s32_relu...................................................................................................................... 400
__vmaxs2................................................................................................................................... 401
__vmaxs4................................................................................................................................... 401
__vmaxu2................................................................................................................................... 401
__vmaxu4................................................................................................................................... 402
__vmins2.................................................................................................................................... 402
__vmins4.................................................................................................................................... 402
__vminu2................................................................................................................................... 402
__vminu4................................................................................................................................... 403
__vneg2...................................................................................................................................... 403
__vneg4...................................................................................................................................... 403
__vnegss2.................................................................................................................................. 404
__vnegss4.................................................................................................................................. 404
__vsads2.................................................................................................................................... 404
__vsads4.................................................................................................................................... 405
__vsadu2.................................................................................................................................... 405
__vsadu4.................................................................................................................................... 405
__vseteq2................................................................................................................................... 406
__vseteq4................................................................................................................................... 406
__vsetges2................................................................................................................................. 406
__vsetges4................................................................................................................................. 407
__vsetgeu2................................................................................................................................. 407
__vsetgeu4................................................................................................................................. 407
__vsetgts2.................................................................................................................................. 408
__vsetgts4.................................................................................................................................. 408
__vsetgtu2.................................................................................................................................. 408
__vsetgtu4.................................................................................................................................. 409
__vsetles2................................................................................................................................... 409
__vsetles4................................................................................................................................... 409
__vsetleu2................................................................................................................................... 410
__vsetleu4................................................................................................................................... 410
__vsetlts2................................................................................................................................... 410
__vsetlts4................................................................................................................................... 411
__vsetltu2................................................................................................................................... 411
__vsetltu4................................................................................................................................... 411
__vsetne2................................................................................................................................... 412
__vsetne4...................................................................................................................................412
__vsub2......................................................................................................................................412
__vsub4......................................................................................................................................413
__vsubss2.................................................................................................................................. 413
__vsubss4.................................................................................................................................. 413
__vsubus2..................................................................................................................................414
__vsubus4..................................................................................................................................414

Chapter 2. Data Structures.............................................................................................. 415
__nv_fp8_e4m3............................................................................................................................. 415
__x..............................................................................................................................................415
__nv_fp8_e4m3......................................................................................................................... 415
__nv_fp8_e4m3......................................................................................................................... 416
__nv_fp8_e4m3......................................................................................................................... 416
__nv_fp8_e4m3......................................................................................................................... 416
__nv_fp8_e4m3......................................................................................................................... 416
__nv_fp8_e4m3......................................................................................................................... 416
__nv_fp8_e4m3......................................................................................................................... 416
__nv_fp8_e4m3......................................................................................................................... 416
__nv_fp8_e4m3......................................................................................................................... 417
__nv_fp8_e4m3......................................................................................................................... 417
__nv_fp8_e4m3......................................................................................................................... 417
__nv_fp8_e4m3......................................................................................................................... 417
__nv_fp8_e4m3......................................................................................................................... 417
operator __half......................................................................................................................... 417
operator __nv_bfloat16.............................................................................................................418
operator bool.............................................................................................................................418
operator double........................................................................................................................ 418
operator float............................................................................................................................ 418
operator int............................................................................................................................... 418
operator long long int.............................................................................................................. 418
operator short int..................................................................................................................... 418
operator signed char................................................................................................................419
operator unsigned char...........................................................................................................419
operator unsigned int.............................................................................................................419
operator unsigned long long int..............................................................................................419
operator unsigned short int.....................................................................................................419
__nv_fp8_e5m2............................................................................................................................. 419
__x..............................................................................................................................................420
__nv_fp8_e5m2......................................................................................................................... 420
__nv_fp8_e5m2......................................................................................................................... 420
__nv_fp8_e5m2......................................................................................................................... 420

CUDA Math API
vRelease Version | xxxi
operator __half2....................................................................................................................... 426
operator float2...................................................................................................................... 426
__nv_fp8x4_e4m3......................................................................................................................427
__x..............................................................................................................................................427
__nv_fp8x4_e4m3......................................................................................................................427
__nv_fp8x4_e4m3......................................................................................................................427
__nv_fp8x4_e4m3......................................................................................................................427
__nv_fp8x4_e4m3......................................................................................................................427
__nv_fp8x4_e4m3......................................................................................................................427
__nv_fp8x4_e4m3......................................................................................................................427
__nv_fp8x4_e4m3......................................................................................................................428
operator float4...................................................................................................................... 428
__nv_fp8x4_e5m2......................................................................................................................428
__x..............................................................................................................................................428
__nv_fp8x4_e5m2......................................................................................................................428
__nv_fp8x4_e5m2......................................................................................................................428
__nv_fp8x4_e5m2......................................................................................................................428
__nv_fp8x4_e5m2......................................................................................................................429
__nv_fp8x4_e5m2......................................................................................................................429
__nv_fp8x4_e5m2......................................................................................................................429
operator float4...................................................................................................................... 429

Chapter 3. Data Fields........................................................................................................430
Chapter 1. Modules

Here is a list of all modules:

- **FP8 Intrinsics**
  - 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 Intrinsics**
  - 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 Intrinsics**
  - Bfloat16 Arithmetic Functions
  - Bfloat162 Arithmetic Functions
  - Bfloat16 Comparison Functions
  - Bfloat162 Comparison Functions
  - Bfloat16 Precision Conversion and Data Movement
1.1. **FP8 Intrinsics**

This section describes fp8 intrinsic functions. To use these functions, include the header file `cuda_fp8.h` in your program.

**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.
_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.

__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__ __half_raw __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__ __half2_raw __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 \( \text{fp8} \) values of the kind specified by \( \text{fp8\_interpretation} \) parameter, using round-to-nearest-even rounding and saturation mode specified by \( \text{saturate} \) parameter.

\[
\text{__host____device____nv\_fp8\_storage\_t}
\]
\[
\text{__nv\_cvt\_halfraw\_to\_fp8}(\text{const \_half\_raw } x, \text{ const \_nv\_saturation\_t saturate, const \_nv\_fp8\_interpretation\_t fp8\_interpretation})
\]

Converts input \( \text{half} \) precision \( x \) to \( \text{fp8} \) type of the requested kind using round-to-nearest-even rounding and requested saturation mode.

**Returns**

- The \( \text{__nv\_fp8\_storage\_t} \) value holds the result of conversion.

**Description**

Converts input \( x \) to \( \text{fp8} \) type of the kind specified by \( \text{fp8\_interpretation} \) parameter, using round-to-nearest-even rounding and saturation mode specified by \( \text{saturate} \) parameter.

### 1.1.2. C++ struct for handling \( \text{fp8} \) data type of \( \text{e5m2} \) kind.

**FP8 Intrinsics**

```cpp
struct __nv\_fp8\_e5m2
\__nv\_fp8\_e5m2\_datatype
\__nv\_fp8\_storage\_t :: \_nv\_fp8\_e5m2::\_x
```

Storage variable contains the \( \text{fp8} \) floating-point data.

```cpp
\text{__host____device____nv\_fp8\_e5m2::\_nv\_fp8\_e5m2\_datatype (const long long int val)}
```

**Description**

Constructor from \( \text{long long int} \) data type, relies on \( \text{__NV\_SATFINITE} \) 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 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 double()

Description
Conversion operator to double data type.

__host____device____nv_fp8_e5m2::operator float()

Description
Conversion operator to float data type.

__host____device____nv_fp8_e5m2::operator int()

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

__host____device____nv_fp8_e5m2::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 0x8000000000000000LL.
__host__ device__nv_fp8_e5m2::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__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 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_bfloat162 f)

Description
Constructor from __nv_bfloat162 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 (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 (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 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 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 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 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

```cpp
struct __nv_fp8x2_e4m3
__nv_fp8x2_e4m3 datatype

__nv_fp8x2_storage_t __nv_fp8x2_e4m3::e4m3::x
```

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

```cpp
__host__ __device__ __nv_fp8x2_e4m3::e4m3::e4m3
(const double2 f)
```

Description

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

```cpp
__host__ __device__ __nv_fp8x2_e4m3::e4m3::e4m3
(const float2 f)
```

Description

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

```cpp
__host__ __device__ __nv_fp8x2_e4m3::e4m3::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_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_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 that are only supported in device code. To use these functions, include the header file `cuda_fp16.h` in your program.

**Half Arithmetic Functions**

**Half2 Arithmetic Functions**

**Half Comparison Functions**

**Half2 Comparison Functions**

**Half Precision Conversion and Data Movement**

**Half Math Functions**

**Half2 Math Functions**

1.2.1. Half Arithmetic Functions

Half Precision Intrinsics

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

```c
__device__ __half __habs (const __half a)
```

Calculates the absolute value of input `half` number and returns the result.

**Parameters**

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

**Returns**

- `half`
  - The absolute value of `a`. 
Description
Calculates the absolute value of input `half` number and returns the result.

```c
__device__ __half __hadd (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.

```c
__device__ __half __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.

```c
__device__ __half __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.
__device__ __half __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 mode.

__device__ __half __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__ __half __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__ __half __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.

**__device__ __half __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 mode.

**__device__ __half __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 mode. Prevents floating-point contractions of mul+add or sub into fma.
**__device__ __half __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 half multiplication of inputs a and b, in round-to-nearest mode, and clamps the result to range [0.0, 1.0]. NaN results are flushed to +0.0.

**__device__ __half __hneg (const __half a)**

Negates input half number and returns the result.

**Description**

Negates input half number and returns the result.

**__device__ __half __hsub (const __half a, const __half b)**

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

**Description**

Subtracts half input b from input a in round-to-nearest mode.

**__device__ __half __hsub_rn (const __half a, const __half b)**

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

**Description**

Subtracts half input b from input a in round-to-nearest mode. Prevents floating-point contractions of mul+sub into fma.
__device__ __half__ __hsub_sat (const __half a, const __half b)

Performs half subtraction 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 subtraction of b from a, with respect to saturation.

Description

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

__device__ __half__ 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.
1.2.2. Half2 Arithmetic Functions

Half Precision Intrinsics

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

`__device__ __half2 __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 mode.

`__device__ __half2 __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.
__device__ __half2 __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 mode.

__device__ __half2 __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 mode. Prevents floating-point contractions of mul+add into fma.

__device__ __half2 __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 mode, and clamps the results to range [0.0, 1.0]. NaN results are flushed to +0.0.
__device__ __half2 __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__ __half2 __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__ __half2 __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__ __half2 __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.
__device__ __half2 __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.

__device__ __half2 __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.

__device__ __half2 __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.
__device__ __half2 __hneg2 (const __half2 a)
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.

__device__ __half2 __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.

__device__ __half2 __hsub2_rn (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.

__device__ __half2 __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 \texttt{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.

\begin{verbatim}
\_device\_ \_half2 atomicAdd (const \_half2 *address, const \_half2 val)
\end{verbatim}

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{half} elements; the entire \texttt{half2} is not guaranteed to be atomic as a single 32-bit access.

Parameters
\begin{itemize}
\item \texttt{address} - \texttt{half2*}. An address in global or shared memory.
\item \texttt{val} - \texttt{half2}. The value to be added.
\end{itemize}

Returns
\begin{itemize}
\item \texttt{half2}
\item The old value read from \texttt{address}.
\end{itemize}

Description
The location of \texttt{address} must be in global or shared memory. This operation has undefined behavior otherwise. This operation is only supported by devices of compute capability 6.x and higher.

\begin{tabular}{|l|}
\hline
\textbf{Note:} \\
For more details for this function see the Atomic Functions section in the CUDA C++ Programming Guide. \\
\hline
\end{tabular}

1.2.3. Half Comparison Functions
Half Precision Intrinsics
To use these functions, include the header file \texttt{cuda_fp16.h} in your program.
__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.

__device__ bool __hequ (const __half a, const __half b)
Performs half unordered if-equal comparison.

Parameters

a
- half. Is only being read.

b
- half. Is only being read.

Returns

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

Description

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

__device__ bool __hge (const __half a, const __half b)
Performs half greater-equal comparison.

Parameters

a
- half. Is only being read.
__device__ bool __hgeu (const __half a, const __half b)
Performs half unordered greater-equal comparison.

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

Returns
bool

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

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

__device__ bool __hgt (const __half a, const __half b)
Performs half greater-than comparison.

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

Returns
bool

The boolean result of greater-equal comparison of a and b.
The boolean result of greater-than comparison of \( a \) and \( b \).

**Description**

Performs half greater-than comparison of inputs \( a \) and \( b \). NaN inputs generate false results.

```c
__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.

```c
__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.
__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.

__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.

__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.

__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.

__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.
__device__ __half __hmax (const __half a, const __half b)
Calculates half maximum of two input values.

Description
Calculates half max(a, b) defined as \((a > b) \cdot a + (b > a) \cdot b\).

‣ 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

__device__ __half __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 \((a > b) \cdot a + (b > a) \cdot b\).

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

__device__ __half __hmin (const __half a, const __half b)
Calculates half minimum of two input values.

Description
Calculates half min(a, b) defined as \((a < b) \cdot a + (b < a) \cdot b\).

‣ 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

__device__ __half __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 \((a < b) \cdot a + (b < a) \cdot b\).

‣ 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
__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.

```c
__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.

### 1.2.4. Half2 Comparison Functions

**Half Precision Intrinsics**

To use these functions, include the header file `cuda_fp16.h` in your program.
__device__ bool __hbeq2 (const __half2 a, const __half2 b)
Performs half2 vector 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 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 false results.

__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.

__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.

__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 \texttt{half} results of unordered greater-equal comparison of vectors \texttt{a} and \texttt{b} are true;

false otherwise.

\textbf{Description}

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

\begin{verbatim}
__device__ bool __hbgt2 (const __half2 a, const __half2 b)

defines the function \texttt{__hbgt2} that performs \texttt{half2} vector greater-equal comparison of inputs \texttt{a} and \texttt{b}. The bool result is set to true only if both \texttt{half} greater-equal comparisons evaluate to true, or false otherwise. NaN inputs generate true results.

\textbf{Parameters}

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

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

\textbf{Returns}

\texttt{bool}

true if both \texttt{half} results of greater-than comparison of vectors \texttt{a} and \texttt{b} are true;

false otherwise.

\textbf{Description}

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

\begin{verbatim}
__device__ bool __hbgtu2 (const __half2 a, const __half2 b)

defines the function \texttt{__hbgtu2} that performs \texttt{half2} vector unordered greater-than comparison of inputs \texttt{a} and \texttt{b}. The bool result is set to true only if both \texttt{half} unordered greater-than comparisons evaluate to true, or false otherwise. NaN inputs generate false results.

\textbf{Parameters}

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

\texttt{b}
- \texttt{half2}. Is only being read.
Returns
bool
▷ true if both \texttt{half} results of unordered greater-than comparison of vectors \texttt{a} and \texttt{b} are true;
▷ false otherwise.

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

\texttt{\_device\_ bool \_hble2 (const \_half2 \texttt{a}, const \_half2 \texttt{b})}
Performs \texttt{half2} vector less-equal comparison and returns boolean true iff both \texttt{half} results are true, boolean false otherwise.

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

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

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

\texttt{\_device\_ bool \_hbleu2 (const \_half2 \texttt{a}, const \_half2 \texttt{b})}
Performs \texttt{half2} vector unordered less-equal comparison and returns boolean true iff both \texttt{half} results are true, boolean false otherwise.

Parameters
\texttt{a}
- \texttt{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.

__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.
__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.

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

\texttt{__device__ bool __hbneu2 (const __half2 a, const __half2 b)}
Performs \texttt{half2} vector unordered not-equal comparison and returns boolean true iff both \texttt{half} results are true, boolean false otherwise.

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

Returns
\texttt{bool}
\begin{itemize}
  \item true if both \texttt{half} results of unordered not-equal comparison of vectors \texttt{a} and \texttt{b} are true;
  \item false otherwise.
\end{itemize}

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

\texttt{__device__ __half2 __heq2 (const __half2 a, const __half2 b)}
Performs \texttt{half2} vector if-equal comparison.

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

Returns
\texttt{half2}
\begin{itemize}
  \item The vector result of if-equal comparison of vectors \texttt{a} and \texttt{b}.
\end{itemize}
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.

__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.

__device__ __half2 __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 \texttt{half2} vector if-equal comparison of inputs \(a\) and \(b\). The corresponding \texttt{half} results are set to 1.0 for true, or 0.0 for false. NaN inputs generate true results.

\begin{verbatim}
__device__ unsigned __hequ2_mask (const __half2 a, const __half2 b)
Performs \texttt{half2} vector unordered if-equal comparison.

**Parameters**

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

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

**Returns**

\texttt{unsigned int}
- The vector mask result of unordered if-equal comparison of vectors \(a\) and \(b\).

**Description**

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

\begin{verbatim}
__device__ __half2 __hge2 (const __half2 a, const __half2 b)
Performs \texttt{half2} vector greater-equal comparison.

**Parameters**

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

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

**Returns**

\texttt{half2}
- The vector result of greater-equal comparison of vectors \(a\) and \(b\).

**Description**

Performs \texttt{half2} vector greater-equal comparison of inputs \(a\) and \(b\). The corresponding \texttt{half} results are set to 1.0 for true, or 0.0 for false. NaN inputs generate false results.
__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.

__device__ __half2 __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.
__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.

__device__ __half2 __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.
__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.

__device__ __half2 __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.
__device__ unsigned __hgtu2_mask (const __half2 a, const __half2 b)
Performs \texttt{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 \texttt{a} and \texttt{b}.

Description
Performs \texttt{half2} 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.

__device__ __half2 __hisnan2 (const __half2 a)
Determine whether \texttt{half2} argument is a NaN.

Parameters
a
- half2. Is only being read.

Returns
half2
- The \texttt{half2} with the corresponding \texttt{half} results set to 1.0 for NaN, 0.0 otherwise.

Description
Determine whether each half of input \texttt{half2} number \texttt{a} is a NaN.

__device__ __half2 __hle2 (const __half2 a, const __half2 b)
Performs \texttt{half2} vector less-equal comparison.

Parameters
a
- 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.

__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.

__device__ __half2 __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 \texttt{half2} vector less-equal comparison of inputs a and b. The corresponding \texttt{half} results are set to 1.0 for true, or 0.0 for false. NaN inputs generate true results.

\texttt{\_\_device\_\_unsigned \_\_hleu2\_\_mask (const \_\_half2 a, const \_\_half2 b)}
Performs \texttt{half2} vector unordered less-equal comparison.

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

Returns
unsigned int

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

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

\texttt{\_\_device\_\_half2 \_\_hlt2 (const \_\_half2 a, const \_\_half2 b)}
Performs \texttt{half2} vector less-than comparison.

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

Returns
half2

- The \texttt{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.

`__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.

`__device__ __half2 __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.

```__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.

```__device__ __half2 __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`
__device__ __half2 __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

__device__ __half2 __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

__device__ __half2 __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
__device__ __half2 __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.

__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.
__device__ __half2 __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.

__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.
1.2.5. 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__ __half __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__ __half2 __float2half2_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 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.
__host__ __device__ __half __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.

__host__ __device__ __half2 __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__ __half __float2half_rd (const float a)
Converts float number to half precision in round-down 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-down mode.

__host__ __device__ __half __float2half_rn (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.

__host__ __device__ __half __float2half_ru (const float a)
Converts float number to half precision in round-up 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-up mode.
__host____device__ __half __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__ __half2 __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.
__host____device__ float2 __half2float2 (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.

__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.

__device__ __half2 __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.

__device__ int __half2int_rn (const __half h)
Convert a half to a signed integer in round-to-nearest-even 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-to-nearest-even mode. NaN inputs are converted to 0.
__device__ int __half2int_ru (const __half h)
Convert a half to a signed integer in round-up 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-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.

__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.

__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.

\[
\textbf{__device__ unsigned int __half2uint_rd (const __half h)}
\]
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.

\[
\textbf{__device__ unsigned int __half2uint_rn (const __half h)}
\]
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.

\[
\text{\_device\_ unsigned long long int __half2ull_rn (const __half h)}
\]
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.

\[
\text{\_device\_ unsigned long long int __half2ull_ru (const __half h)}
\]
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.

```c
__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.

```c
__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

‣ 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

‣ 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.

__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.

__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.

\[ \text{\_device\_\_half2 \_halves2half2 (const \_half a, const \_half b)} \]
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.

\[ \text{\_host\_\_device\_\_float \_high2float (const \_half2 a)} \]
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.

__device__ __half __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.

__device__ __half2 __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 half2 input a and returns a new half2 number which has both halves equal to the extracted bits.
\texttt{\_\_device\_ \_half2 \_\_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**

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

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

**Returns**

\texttt{half2}

\begin{itemize}
  \item 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{\_\_device\_ \_half \_\_int2half_rd (const int i)}

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

**Parameters**

\texttt{i}
- \texttt{int}. Is only being read.

**Returns**

\texttt{half}

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

**Description**

Convert the signed integer value \texttt{i} to a half-precision floating-point value in round-down mode.
__host__ __device__ __half __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.

__device__ __half __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.

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

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

Parameters
ptr
- memory location

Returns
The value pointed by `ptr`

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

Parameters
ptr
- memory location

Returns
The value pointed by `ptr`

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

Parameters
ptr
- memory location

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

Parameters
ptr  
- memory location

Returns
The value pointed by `ptr`

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

Parameters
ptr  
- memory location

Returns
The value pointed by `ptr`

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

Parameters
ptr  
- memory location

Returns
The value pointed by `ptr`

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

Parameters
ptr  
- memory location

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

Parameters
ptr
- memory location

Returns
The value pointed by `ptr`

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

Parameters
ptr
- memory location

Returns
The value pointed by `ptr`

__device__ __half2 __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__ __half __ldlu (const __half *ptr)
Generates a `ld.global.lu` load instruction.

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

Parameters

ptr
- memory location

Returns
The value pointed by `ptr`

__device__ __half __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__ __half __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.

\[
\text{__device__ __half __ll2half_ru (const long long int } i)\
\]
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.

\[
\text{__device__ __half __ll2half_rz (const long long int } i)\
\]
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.

__device__ __half__ __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.

__device__ __half2__ __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 half2 input a and returns a new half2 number which has both halves equal to the extracted bits.

__device__ __half2 __lows2half2 (const __half2 a, const __half2 b)
Extracts low 16 bits from each of the two half2 inputs and combines into one half2 number.

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

Returns
half2
▶ The low 16 bits of a and of b.

Description
Extracts low 16 bits from each of the two half2 inputs and combines into one half2 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.
__device__ __half __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__ __half2 __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__ __half __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__ __half2 __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__ __half __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.

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

__device__ __half2 __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.

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

```cpp
__device__ __half __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__ __half2 __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**
  - halfl. 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 halfl. 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__ __half __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__ __half __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.

__device__ __half __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.

```cpp
__device__ __half __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.

```cpp
__device__ __half __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
__device__ __half __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.

__host____device__ __half __uint2half_rn (const unsigned int i)
Convert an unsigned integer to a half in round-to-nearest-even 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-to-nearest-even mode.

__device__ __half __uint2half_ru (const unsigned int i)
Convert an unsigned integer to a half in round-up 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-up mode.

\[
\text{\texttt{__device__ \texttt{\_half \_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.

\[
\text{\texttt{__device__ \texttt{\_half \_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__ __half __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.

__device__ __half __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.
__device__ __half __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.

__device__ __half __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__ __half __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.

__device__ __half __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.
\_\_device\_\_ \_half \_\_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.

\_\_device\_\_ \_half \_\_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.

1.2.6. Half Math Functions

Half Precision Intrinsics

To use these functions, include the header file cuda\_fp16.h in your program.
__device__ __half hceil (const __half h)
Calculate ceiling of the input argument.

Parameters
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__ __half hcos (const __half a)
Calculates __half cosine in round-to-nearest-even mode.

Parameters
a
  - half. Is only being read.

Returns
half
  ▶ The cosine of a.

Description
Calculates __half cosine of input a in round-to-nearest-even mode.

__device__ __half hexp (const __half a)
Calculates __half natural exponential function in round-to-nearest mode.

Parameters
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__ __half hexp10 (const __half a)
Calculates half decimal exponential function in round-to-nearest 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__ __half hexp2 (const __half a)
Calculates half binary exponential function in round-to-nearest 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__ __half 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__ __half 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__ __half 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__ __half 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__ __half 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__ __half 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__ __half hsqrt (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 mode.

__device__ __half 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__ __half 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__ __half 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.

1.2.7. Half2 Math Functions
Half Precision Intrinsics
To use these functions, include the header file cuda_fp16.h in your program.
__device__ __half2 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__ __half2 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__ __half2 h2exp (const __half2 a)
Calculates half2 vector exponential function in round-to-nearest mode.

Parameters
a
- half2. Is only being read.

Returns
half2
- The elementwise exponential function on vector a.
Description
Calculates half2 exponential function of input vector a in round-to-nearest-even mode.

__device__ __half2 h2exp10 (const __half2 a)
Calculates half2 vector decimal exponential function in round-to-nearest-even mode.

Parameters
a
- half2. Is only being read.

Returns
half2
- The elementwise decimal exponential function on vector a.

Description
Calculates half2 decimal exponential function of input vector a in round-to-nearest-even mode.

__device__ __half2 h2exp2 (const __half2 a)
Calculates half2 vector binary exponential function in round-to-nearest-even mode.

Parameters
a
- half2. Is only being read.

Returns
half2
- The elementwise binary exponential function on vector a.

Description
Calculates half2 binary exponential function of input vector a in round-to-nearest-even mode.
__device__ __half2 h2floor (const __half2 h)

Calculate the largest integer less than or equal to \( h \).

**Parameters**

\( h \)
- half2. Is only being read.

**Returns**

\( \text{half2} \)

- 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__ __half2 h2log (const __half2 a)

Calculates half2 vector natural logarithm in round-to-nearest-even mode.

**Parameters**

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

**Returns**

\( \text{half2} \)

- The elementwise natural logarithm on vector \( a \).

**Description**

Calculates half2 natural logarithm of input vector \( a \) in round-to-nearest-even mode.

__device__ __half2 h2log10 (const __half2 a)

Calculates half2 vector decimal logarithm in round-to-nearest-even mode.

**Parameters**

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

**Returns**

\( \text{half2} \)
The elementwise decimal logarithm on vector a.

**Description**

Calculates \texttt{half2} decimal logarithm of input vector \(a\) in round-to-nearest-even mode.

\[
\texttt{__device__ __half2 h2log2 (const __half2 a)}
\]

Calculates \texttt{half2} vector binary logarithm in round-to-nearest-even mode.

**Parameters**

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

**Returns**

\texttt{half2}

- The elementwise binary logarithm on vector \(a\).

**Description**

Calculates \texttt{half2} binary logarithm of input vector \(a\) in round-to-nearest mode.

\[
\texttt{__device__ __half2 h2rcp (const __half2 a)}
\]

Calculates \texttt{half2} vector reciprocal in round-to-nearest-even mode.

**Parameters**

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

**Returns**

\texttt{half2}

- The elementwise reciprocal on vector \(a\).

**Description**

Calculates \texttt{half2} reciprocal of input vector \(a\) in round-to-nearest-even mode.
__device__ __half2 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__ __half2 h2rsqrt (const __half2 a)
Calculates half2 vector reciprocal square root in round-to-nearest 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__ __half2 h2sin (const __half2 a)
Calculates half2 vector sine in round-to-nearest-even mode.

Parameters
a
- half2. Is only being read.

Returns
half2
The elementwise sine on vector \( a \).

**Description**
Calculates \( \text{half2} \) sine of input vector \( a \) in round-to-nearest-even mode.

\[
\text{__device__ __half2 h2sqrt (const __half2 a)}
\]
Calculates \( \text{half2} \) vector square root in round-to-nearest-even mode.

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

**Returns**
half2
- The elementwise square root on vector \( a \).

**Description**
Calculates \( \text{half2} \) square root of input vector \( a \) in round-to-nearest mode.

\[
\text{__device__ __half2 h2trunc (const __half2 h)}
\]
Truncate \( \text{half2} \) vector input argument to the integral part.

**Parameters**
\( h \)
- half2. Is only being read.

**Returns**
half2
- The truncated \( h \).

**Description**
Round each component of vector \( h \) to the nearest integer value that does not exceed \( h \) in magnitude.

### 1.3. Bfloat16 Precision Intrinsics

This section describes \( \text{nv_bfloat16} \) precision intrinsic functions that are only supported in device code. To use these functions, include the header file \texttt{cuda_bf16.h} in your program.
Bfloat16 Arithmetic Functions

Bfloat162 Arithmetic Functions

Bfloat16 Comparison Functions

Bfloat162 Comparison Functions

Bfloat16 Precision Conversion and Data Movement

Bfloat16 Math Functions

Bfloat162 Math Functions

1.3.1. Bfloat16 Arithmetic Functions

Bfloat16 Precision Intrinsics

To use these functions, include the header file cuda_bf16.h in your program.

__device__ __nv_bfloat162 __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 mode.

__device__ __nv_bfloat16 __fabs (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.

```c
__device__ __nv_bfloat16 __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.

```c
__device__ __nv_bfloat16 __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.

```c
__device__ __nv_bfloat16 __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 `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.
__device__ __nv_bfloat16 __hdiv (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs \texttt{nv\_bfloat16} division in round-to-nearest-even mode.

Description
Divides \texttt{nv\_bfloat16} input \texttt{a} by input \texttt{b} in round-to-nearest mode.

__device__ __nv_bfloat16 __hfma (const __nv_bfloat16 a, const __nv_bfloat16 b, const __nv_bfloat16 c)
Performs \texttt{nv\_bfloat16} fused multiply-add in round-to-nearest-even mode.

Description
Performs \texttt{nv\_bfloat16} multiply on inputs \texttt{a} and \texttt{b}, then performs a \texttt{nv\_bfloat16} add of the result with \texttt{c}, rounding the result once in round-to-nearest-even mode.

__device__ __nv_bfloat16 __hfma\_relu (const __nv_bfloat16 a, const __nv_bfloat16 b, const __nv_bfloat16 c)
Performs \texttt{nv\_bfloat16} fused multiply-add in round-to-nearest-even mode with relu saturation.

Parameters
\texttt{a}
- \texttt{nv\_bfloat16}. Is only being read.

\texttt{b}
- \texttt{nv\_bfloat16}. Is only being read.

\texttt{c}
- \texttt{nv\_bfloat16}. Is only being read.

Returns
\texttt{nv\_bfloat16}

- The result of fused multiply-add operation on \texttt{a}, \texttt{b}, and \texttt{c} with relu saturation.

Description
Performs \texttt{nv\_bfloat16} multiply on inputs \texttt{a} and \texttt{b}, then performs a \texttt{nv\_bfloat16} add of the result with \texttt{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__ __nv_bfloat16 __hfma_sat (const __nv_bfloat16 a, const __nv_bfloat16 b, const __nv_bfloat16 c)

Performs \texttt{nv\_bfloat16} fused multiply-add in round-to-nearest-even mode, with saturation to \([0.0, 1.0]\).

**Parameters**

- \texttt{a} - \texttt{nv\_bfloat16}. Is only being read.
- \texttt{b} - \texttt{nv\_bfloat16}. Is only being read.
- \texttt{c} - \texttt{nv\_bfloat16}. Is only being read.

**Returns**

\texttt{nv\_bfloat16}

\textgreater; The result of fused multiply-add operation on \(a\), \(b\), and \(c\), with respect to saturation.

**Description**

Performs \texttt{nv\_bfloat16} multiply on inputs \(a\) and \(b\), then performs a \texttt{nv\_bfloat16} 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.

__device__ __nv_bfloat16 __hmul (const __nv_bfloat16 a, const __nv_bfloat16 b)

Performs \texttt{nv\_bfloat16} multiplication in round-to-nearest-even mode.

**Description**

Performs \texttt{nv\_bfloat16} multiplication of inputs \(a\) and \(b\), in round-to-nearest mode.

__device__ __nv_bfloat16 __hmul\_rn (const __nv_bfloat16 a, const __nv_bfloat16 b)

Performs \texttt{nv\_bfloat16} multiplication in round-to-nearest-even mode.

**Description**

Performs \texttt{nv\_bfloat16} multiplication of inputs \(a\) and \(b\), in round-to-nearest mode. Prevents floating-point contractions of mul+add or sub into fma.
__device__ __nv_bfloat16 __hmul_sat (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs \texttt{nv\_bfloat16} multiplication in round-to-nearest-even mode, with saturation to [0.0, 1.0].

Parameters

\texttt{a}
- \texttt{nv\_bfloat16}. Is only being read.

\texttt{b}
- \texttt{nv\_bfloat16}. Is only being read.

Returns
\texttt{nv\_bfloat16}
- The result of multiplying \texttt{a} and \texttt{b}, with respect to saturation.

Description
Performs \texttt{nv\_bfloat16} multiplication of inputs \texttt{a} and \texttt{b}, in round-to-nearest mode, and clamps the result to range [0.0, 1.0]. NaN results are flushed to +0.0.

__device__ __nv_bfloat16 __hneg (const __nv_bfloat16 a)
Negates input \texttt{nv\_bfloat16} number and returns the result.

Description
Negates input \texttt{nv\_bfloat16} number and returns the result.

__device__ __nv_bfloat16 __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 \texttt{b} from input \texttt{a} in round-to-nearest mode.
__device__ __nv_bfloat16 __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 \texttt{b} from input \texttt{a} in round-to-nearest mode. Prevents floating-point contractions of mul+sub into fma.

__device__ __nv_bfloat16 __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**

\texttt{a}
- \texttt{nv\_bfloat16}. Is only being read.

\texttt{b}
- \texttt{nv\_bfloat16}. Is only being read.

**Returns**

\texttt{nv\_bfloat16}

- The result of subtraction of \texttt{b} from \texttt{a}, with respect to saturation.

**Description**

Subtracts \texttt{nv\_bfloat16} input \texttt{b} from input \texttt{a} in round-to-nearest mode, and clamps the result to range \([0.0, 1.0]\). NaN results are flushed to +0.0.

__device__ __nv_bfloat16 atomicAdd (const __nv_bfloat16 *address, const __nv_bfloat16 val)

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

**Parameters**

\texttt{address}
- \texttt{__nv\_bfloat16*}. An address in global or shared memory.

\texttt{val}
- \texttt{__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 only supported by devices of compute capability 8.x and higher.

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

1.3.2. Bfloat162 Arithmetic Functions
Bfloat162 Precision Intrinsics
To use these functions, include the header file cuda_bf16.h in your program.

__device__ __nv_bfloat162 __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.
__device__ __nv_bfloat162 __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 mode.

__device__ __nv_bfloat162 __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 mode. Prevents floating-point contractions of mul+add into fma.

__device__ __nv_bfloat162 __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 mode, and clamps the results to range [0.0, 1.0]. NaN results are flushed to +0.0.
__device__ __nv_bfloat162 __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__ __nv_bfloat162 __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__ __nv_bfloat162 __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.
 Returns
nv_bfloat162

<table>
<thead>
<tr>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>nv_bfloat162</td>
</tr>
</tbody>
</table>

Description
Performs \( \text{nv}_\text{bfloat162} \) vector multiply on inputs \( a \) and \( b \), then performs a \( \text{nv}_\text{bfloat162} \) 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.

\[
\text{__device__ __nv_bfloat162 __hfma2_sat (const __nv_bfloat162 a, const __nv_bfloat162 b, const __nv_bfloat162 c)}
\]

Performa
\( \text{nv}_\text{bfloat162} \) vector fused multiply-add in round-to-nearest-even mode, with saturation to \([0.0, 1.0]\).

Parameters
\( a \)

- \( \text{nv}_\text{bfloat162} \). Is only being read.

\( b \)

- \( \text{nv}_\text{bfloat162} \). Is only being read.

\( c \)

- \( \text{nv}_\text{bfloat162} \). Is only being read.

Returns
nv_bfloat162

<table>
<thead>
<tr>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>nv_bfloat162</td>
</tr>
</tbody>
</table>

Description
Performs \( \text{nv}_\text{bfloat162} \) vector multiply on inputs \( a \) and \( b \), then performs a \( \text{nv}_\text{bfloat162} \) 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.
__device__ __nv_bfloat162 __hmul2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs \texttt{nv\_bfloat162} vector multiplication in round-to-nearest-even mode.

Description
Performs \texttt{nv\_bfloat162} vector multiplication of inputs \texttt{a} and \texttt{b}, in round-to-nearest-even mode.

__device__ __nv_bfloat162 __hmul2\_rn (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs \texttt{nv\_bfloat162} vector multiplication in round-to-nearest-even mode.

Description
Performs \texttt{nv\_bfloat162} vector multiplication of inputs \texttt{a} and \texttt{b}, in round-to-nearest-even mode. Prevents floating-point contractions of \texttt{mul+add} or \texttt{sub} into \texttt{fma}.

__device__ __nv_bfloat162 __hmul2\_sat (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs \texttt{nv\_bfloat162} vector multiplication 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 result of elementwise multiplication of vectors \texttt{a} and \texttt{b}, with respect to saturation.

Description
Performs \texttt{nv\_bfloat162} vector multiplication of inputs \texttt{a} and \texttt{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__ __nv_bfloat162 __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.

__device__ __nv_bfloat162 __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.

__device__ __nv_bfloat162 __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.

__device__ __nv_bfloat162 __hsub2_sat (const __nv_bfloat162 a, const __nv_bfloat162 b)

Performs nv_bfloat162 vector subtraction 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 subtraction of vector b from a, with respect to saturation.
Description

Subtracts \texttt{nv\_bf\textfloatsep162} 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.

\begin{verbatim}
__device__ __nv_bfloat162 atomicAdd (const __nv_bfloat162 *address, const __nv_bfloat162 val)
\end{verbatim}

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 \texttt{nv\_bf\textfloatsep16} elements; the entire \texttt{__nv\_bf\textfloatsep162} is not guaranteed to be atomic as a single 32-bit access.

Parameters

\begin{description}
\item[address] __nv\_bf\textfloatsep162*. An address in global or shared memory.
\item[val] __nv\_bf\textfloatsep162. The value to be added.
\end{description}

Returns

\texttt{__nv\_bf\textfloatsep162}

\begin{itemize}
\item The old value read from \(address\).
\end{itemize}

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 8.x and higher.

Note:

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

1.3.3. Bfloat16 Comparison Functions

Bfloat16 Precision Intrinsics

To use these functions, include the header file \texttt{cuda\_bf\textfloatsep16.h} in your program.
__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.

__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.
__device__ bool __hge (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs \texttt{nv\_bfloat16} greater-equal comparison.

Parameters
\textbf{a}
- \texttt{nv\_bfloat16}. Is only being read.
\textbf{b}
- \texttt{nv\_bfloat16}. Is only being read.

Returns
\texttt{bool}

\begin{itemize}
  \item The boolean result of greater-equal comparison of \texttt{a} and \texttt{b}.
\end{itemize}

Description
Performs \texttt{nv\_bfloat16} greater-equal comparison of inputs \texttt{a} and \texttt{b}. NaN inputs generate false results.

__device__ bool __hgeu (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs \texttt{nv\_bfloat16} unordered greater-equal comparison.

Parameters
\textbf{a}
- \texttt{nv\_bfloat16}. Is only being read.
\textbf{b}
- \texttt{nv\_bfloat16}. Is only being read.

Returns
\texttt{bool}

\begin{itemize}
  \item The boolean result of unordered greater-equal comparison of \texttt{a} and \texttt{b}.
\end{itemize}

Description
Performs \texttt{nv\_bfloat16} greater-equal comparison of inputs \texttt{a} and \texttt{b}. NaN inputs generate true results.
__device__ bool __hgt (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs \texttt{nv\_bf\texttt{o}loat16} greater-than comparison.

Parameters
\begin{itemize}
\item \texttt{a} - \texttt{nv\_bf\texttt{o}loat16}. Is only being read.
\item \texttt{b} - \texttt{nv\_bf\texttt{o}loat16}. Is only being read.
\end{itemize}

Returns
bool
\begin{itemize}
\item The boolean result of greater-than comparison of \texttt{a} and \texttt{b}.
\end{itemize}

Description
Performs \texttt{nv\_bf\texttt{o}loat16} greater-than comparison of inputs \texttt{a} and \texttt{b}. NaN inputs generate false results.

__device__ bool __hgtu (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs \texttt{nv\_bf\texttt{o}loat16} unordered greater-than comparison.

Parameters
\begin{itemize}
\item \texttt{a} - \texttt{nv\_bf\texttt{o}loat16}. Is only being read.
\item \texttt{b} - \texttt{nv\_bf\texttt{o}loat16}. Is only being read.
\end{itemize}

Returns
bool
\begin{itemize}
\item The boolean result of unordered greater-than comparison of \texttt{a} and \texttt{b}.
\end{itemize}

Description
Performs \texttt{nv\_bf\texttt{o}loat16} greater-than comparison of inputs \texttt{a} and \texttt{b}. NaN inputs generate true results.
__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.

__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.

__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 \texttt{nv\_bfloat16} less-equal comparison of inputs a and b. NaN inputs generate false results.

\texttt{\_\_device\_\_bool \_\_hleu (const \_\_nv\_bfloat16 a, const \_\_nv\_bfloat16 b)}
Performs \texttt{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 \texttt{nv\_bfloat16} less-equal comparison of inputs a and b. NaN inputs generate true results.

\texttt{\_\_device\_\_bool \_\_hlt (const \_\_nv\_bfloat16 a, const \_\_nv\_bfloat16 b)}
Performs \texttt{nv\_bfloat16} less-than comparison.

Parameters
a
  - nv_bfloat16. Is only being read.
b
  - nv_bfloat16. Is only being read.
Returns
void
- 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.

_device__ bool __hltu (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs nv_bfloat16 unordered less-than comparison.

Parameters
a
- nv_bfloat16. Is only being read.
b
- nv_bfloat16. Is only being read.

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

Description
Performs nv_bfloat16 less-than comparison of inputs a and b. NaN inputs generate true results.

_device__ __nv_bfloat16 __hmax (const __nv_bfloat16 a, const __nv_bfloat16 b)
Calculates nv_bfloat16 maximum of two input values.

Description
Calculates 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
__device__ __nv_bfloat16 __hmax_nan (const __nv_bfloat16 a, const __nv_bfloat16 b)
Calculates `nv_bfloat16` maximum of two input values, NaNs pass through.

Description
Calculates `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

__device__ __nv_bfloat16 __hmin (const __nv_bfloat16 a, const __nv_bfloat16 b)
Calculates `nv_bfloat16` minimum of two input values.

Description
Calculates `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

__device__ __nv_bfloat16 __hmin_nan (const __nv_bfloat16 a, const __nv_bfloat16 b)
Calculates `nv_bfloat16` minimum of two input values, NaNs pass through.

Description
Calculates `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

__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 `nv_bfloat16` not-equal comparison of inputs `a` and `b`. NaN inputs generate false results.

```c
__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.

### 1.3.4. Bfloat16 Comparison Functions

**Bfloat16 Precision Intrinsics**

To use these functions, include the header file `cuda_bf16.h` in your program.
__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.

__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\_bf\_float16} vector if-equal comparison of inputs \texttt{a} and \texttt{b}. The bool result is set to true only if both \texttt{nv\_bf\_float16} if-equal comparisons evaluate to true, or false otherwise. NaN inputs generate true results.

\textbf{__device__ bool __hbge2 (const __nv\_bf\_float162 a, const __nv\_bf\_float162 b)}
Performs \texttt{nv\_bf\_float16} vector greater-equal comparison and returns boolean true iff both \texttt{nv\_bf\_float16} results are true, boolean false otherwise.

Parameters
\begin{itemize}
  \item \texttt{a}
    \begin{itemize}
      \item - \texttt{nv\_bf\_float16}. Is only being read.
    \end{itemize}
  \item \texttt{b}
    \begin{itemize}
      \item - \texttt{nv\_bf\_float16}. Is only being read.
    \end{itemize}
\end{itemize}

Returns
bool
\begin{itemize}
  \item true if both \texttt{nv\_bf\_float16} results of greater-equal comparison of vectors \texttt{a} and \texttt{b} are true;
  \item false otherwise.
\end{itemize}

Description
Performs \texttt{nv\_bf\_float16} vector greater-equal comparison of inputs \texttt{a} and \texttt{b}. The bool result is set to true only if both \texttt{nv\_bf\_float16} greater-equal comparisons evaluate to true, or false otherwise. NaN inputs generate false results.

\textbf{__device__ bool __hbgeu2 (const __nv\_bf\_float162 a, const __nv\_bf\_float162 b)}
Performs \texttt{nv\_bf\_float16} vector unordered greater-equal comparison and returns boolean true iff both \texttt{nv\_bf\_float16} results are true, boolean false otherwise.

Parameters
\begin{itemize}
  \item \texttt{a}
    \begin{itemize}
      \item - \texttt{nv\_bf\_float16}. Is only being read.
    \end{itemize}
  \item \texttt{b}
    \begin{itemize}
      \item - \texttt{nv\_bf\_float16}. Is only being read.
    \end{itemize}
\end{itemize}
Returns

bool

▫️ true if both nv_bfloat16 results of unordered greater-equal comparison of vectors \( a \) and \( b \) are true;
▫️ false otherwise.

Description

Performs \( \text{nv}_\text{bfloat162} \) vector greater-equal comparison of inputs \( a \) and \( b \). The bool result is set to true only if both \( \text{nv}_\text{bfloat16} \) greater-equal comparisons evaluate to true, or false otherwise. NaN inputs generate true results.

\[ \text{__device__ bool __hbgt2 (const __nv_bfloat162 a, const __nv_bfloat162 b)} \]

Performs \( \text{nv}_\text{bfloat162} \) vector greater-than comparison and returns boolean true iff both \( \text{nv}_\text{bfloat16} \) results are true, boolean false otherwise.

Parameters

\( a \)

- \( \text{nv}_\text{bfloat162} \). Is only being read.

\( b \)

- \( \text{nv}_\text{bfloat162} \). Is only being read.

Returns

bool

▫️ true if both \( \text{nv}_\text{bfloat162} \) results of greater-than comparison of vectors \( a \) and \( b \) are true;
▫️ false otherwise.

Description

Performs \( \text{nv}_\text{bfloat162} \) vector greater-than comparison of inputs \( a \) and \( b \). The bool result is set to true only if both \( \text{nv}_\text{bfloat16} \) greater-than comparisons evaluate to true, or false otherwise. NaN inputs generate false results.
__device__ bool __hbgtu2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs \texttt{nv\_bfloat162} vector unordered greater-than comparison and returns boolean true iff both \texttt{nv\_bfloat16} results are true, boolean false otherwise.

Parameters
a
- \texttt{nv\_bfloat162}. Is only being read.
b
- \texttt{nv\_bfloat162}. Is only being read.

Returns
bool
- true if both \texttt{nv\_bfloat16} results of unordered greater-than comparison of vectors \texttt{a} and \texttt{b} are true;
- false otherwise.

Description
Performs \texttt{nv\_bfloat162} vector greater-than comparison of inputs \texttt{a} and \texttt{b}. The bool result is set to true only if both \texttt{nv\_bfloat16} greater-than comparisons evaluate to true, or false otherwise. NaN inputs generate true results.

__device__ bool __hble2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs \texttt{nv\_bfloat162} vector less-equal comparison and returns boolean true iff both \texttt{nv\_bfloat16} results are true, boolean false otherwise.

Parameters
a
- \texttt{nv\_bfloat162}. Is only being read.
b
- \texttt{nv\_bfloat162}. Is only being read.

Returns
bool
- true if both \texttt{nv\_bfloat16} results of less-equal comparison of vectors \texttt{a} and \texttt{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}
__device__ bool __hbleu2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
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
\(a\)
- \texttt{nv\_bfloat162}. Is only being read.
\(b\)
- \texttt{nv\_bfloat162}. Is only being read.

Returns
bool
- true if both \texttt{nv\_bfloat16} results of unordered less-equal comparison of vectors \(a\) and \(b\) are true;
- false otherwise.
\end{verbatim}

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}
__device__ bool __hblt2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs \texttt{nv\_bfloat162} vector less-than comparison and returns boolean true iff both \texttt{nv\_bfloat16} results are true, boolean false otherwise.

Parameters
\(a\)
- \texttt{nv\_bfloat162}. Is only being read.
\(b\)
- \texttt{nv\_bfloat162}. Is only being read.
\end{verbatim}
Returns
bool
▶ true if both \texttt{nv\_bfloat16} results of less-than comparison of vectors \texttt{a} and \texttt{b} are true;
▶ false otherwise.

Description
Performs \texttt{nv\_bfloat16} vector less-than comparison of inputs \texttt{a} and \texttt{b}. The bool result is set to true only if both \texttt{nv\_bfloat16} less-than comparisons evaluate to true, or false otherwise. NaN inputs generate false results.

\texttt{\_device\_bool\_hbltu2 (const \_nv\_bfloat162 \texttt{a}, const \_nv\_bfloat162 \texttt{b})}
Performs \texttt{nv\_bfloat16} vector unordered less-than comparison and returns boolean true iff both \texttt{nv\_bfloat16} results are true, boolean false otherwise.

Parameters
\texttt{a}
▶ \texttt{nv\_bfloat162}. Is only being read.
\texttt{b}
▶ \texttt{nv\_bfloat162}. Is only being read.

Returns
bool
▶ true if both \texttt{nv\_bfloat16} results of unordered less-than comparison of vectors \texttt{a} and \texttt{b} are true;
▶ false otherwise.

Description
Performs \texttt{nv\_bfloat16} vector less-than comparison of inputs \texttt{a} and \texttt{b}. The bool result is set to true only if both \texttt{nv\_bfloat16} less-than comparisons evaluate to true, or false otherwise. NaN inputs generate true results.
__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.

__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 `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 true results.

`__device__ __nv_bfloat162 __heq2 (const __nv_bfloat162 a, const __nv_bfloat162 b)`
Performs `nv_bfloat162` vector 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 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 false results.

`__device__ unsigned __heq2_mask (const __nv_bfloat162 a, const __nv_bfloat162 b)`
Performs `nv_bfloat162` vector 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 if-equal comparison of vectors `a` and `b`. 
Description
Performs nv_bfloat16 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.

__device__ __nv_bfloat162 __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.

__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.

__device__ __nv_bfloat162 __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.

__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_bfloat162` vector greater-equal comparison of inputs `a` and `b`. The corresponding `unsigned` bits are set to 0xFFFE for true, or 0x0 for false. NaN inputs generate false results.

```c
__device__ __nv_bfloat162 __hgeu2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
```

Performs `__nv_bfloat162` vector unordered greater-equal 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-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 true results.

```c
__device__ unsigned __hgeu2_mask (const __nv_bfloat162 a, const __nv_bfloat162 b)
```

Performs `__nv_bfloat162` vector unordered 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 unordered greater-equal comparison of vectors `a` and `b`. 
Description
Performs \texttt{nv\_bf\_float162} 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.

\begin{verbatim}
__device__ __nv\_bf\_float162 __hgt2 (const __nv\_bf\_float162 a, const __nv\_bf\_float162 b)
Performs \texttt{nv\_bf\_float162} vector greater-than comparison.
\end{verbatim}

Parameters
\(a\)
- \texttt{nv\_bf\_float162}. Is only being read.

\(b\)
- \texttt{nv\_bf\_float162}. Is only being read.

Returns
\texttt{nv\_bf\_float162}
- The vector result of greater-than comparison of vectors \(a\) and \(b\).

Description
Performs \texttt{nv\_bf\_float162} vector greater-than comparison of inputs \(a\) and \(b\). The corresponding \texttt{nv\_bf\_float16} results are set to 1.0 for true, or 0.0 for false. NaN inputs generate false results.

\begin{verbatim}
__device__ unsigned __hgt2\_mask (const __nv\_bf\_float162 a, const __nv\_bf\_float162 b)
Performs \texttt{nv\_bf\_float162} vector greater-than comparison.
\end{verbatim}

Parameters
\(a\)
- \texttt{nv\_bf\_float162}. Is only being read.

\(b\)
- \texttt{nv\_bf\_float162}. Is only being read.

Returns
\texttt{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 0xFFF for true, or 0x0 for false. NaN inputs generate false results.

```
__device__ __nv_bfloat162 __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.

```
__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\_bf\_float162} 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.

\texttt{__device\_\_\_nv\_bf\_float162 \_\_hisnan2 (const \_\_nv\_bf\_float162 \_\_a)\n}\nDetermine whether \texttt{nv\_bf\_float162} argument is a NaN.

Parameters
\(a\)
- \texttt{nv\_bf\_float162}. Is only being read.

Returns
\texttt{nv\_bf\_float162}
- The \texttt{nv\_bf\_float162} with the corresponding \texttt{nv\_bf\_float16} results set to 1.0 for NaN, 0.0 otherwise.

Description
Determine whether each \texttt{nv\_bf\_float16} of input \texttt{nv\_bf\_float162} number \(a\) is a NaN.

\texttt{__device\_\_\_nv\_bf\_float162 \_\_hle2 (const \_\_nv\_bf\_float162 \_\_a, const \_\_nv\_bf\_float162 \_\_b)\n}\nPerforms \texttt{nv\_bf\_float162} vector less-equal comparison.

Parameters
\(a\)
- \texttt{nv\_bf\_float162}. Is only being read.
\(b\)
- \texttt{nv\_bf\_float162}. Is only being read.

Returns
\texttt{nv\_bf\_float162}
- The \texttt{nv\_bf\_float162} result of less-equal comparison of vectors \(a\) and \(b\).
**Description**

Performs \texttt{nv\_bf\_float162} vector less-equal comparison of inputs \(a\) and \(b\). The corresponding \texttt{nv\_bf\_float16} results are set to 1.0 for true, or 0.0 for false. NaN inputs generate false results.

\texttt{\_\_device\_\_\_unsigned \_\_hle2\_\_\_mask (const \_\_nv\_bf\_\_float162 a, const \_\_nv\_bf\_\_float162 b)}

Performs \texttt{nv\_bf\_float162} vector less-equal comparison.

**Parameters**

- \(a\) - \texttt{nv\_bf\_float162}. Is only being read.
- \(b\) - \texttt{nv\_bf\_float162}. Is only being read.

**Returns**

- unsigned int

\[\text{The vector mask result of less-equal comparison of vectors } a \text{ and } b.\]

**Description**

Performs \texttt{nv\_bf\_float162} 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.

\texttt{\_\_device\_\_\_\_nv\_bf\_\_float162 \_\_hleu2 (const \_\_nv\_bf\_\_float162 a, const \_\_nv\_bf\_\_float162 b)}

Performs \texttt{nv\_bf\_float162} vector unordered less-equal comparison.

**Parameters**

- \(a\) - \texttt{nv\_bf\_float162}. Is only being read.
- \(b\) - \texttt{nv\_bf\_float162}. Is only being read.

**Returns**

- \texttt{nv\_bf\_float162}

\[\text{The vector result of unordered less-equal comparison of vectors } a \text{ and } b.\]
Description
Performs \texttt{nv\_bf\_float162} vector less-equal comparison of inputs \(a\) and \(b\). The corresponding \texttt{nv\_bf\_float16} results are set to 1.0 for true, or 0.0 for false. NaN inputs generate true results.

\begin{verbatim}
__device__ unsigned __hleu2_mask (const __nv\_bf\_float162 a, const __nv\_bf\_float162 b)
\end{verbatim}
Performs \texttt{nv\_bf\_float162} vector unordered less-equal comparison.

Parameters
\begin{itemize}
\item \(a\) - \texttt{nv\_bf\_float162}. Is only being read.
\item \(b\) - \texttt{nv\_bf\_float162}. Is only being read.
\end{itemize}

Returns
unsigned int
\begin{itemize}
\item The vector mask result of unordered less-equal comparison of vectors \(a\) and \(b\).
\end{itemize}

Description
Performs \texttt{nv\_bf\_float162} 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.

\begin{verbatim}
__device__ __nv\_bf\_float162 \_\_hlt2 (const __nv\_bf\_float162 a, const __nv\_bf\_float162 b)
\end{verbatim}
Performs \texttt{nv\_bf\_float162} vector less-than comparison.

Parameters
\begin{itemize}
\item \(a\) - \texttt{nv\_bf\_float162}. Is only being read.
\item \(b\) - \texttt{nv\_bf\_float162}. Is only being read.
\end{itemize}

Returns
\texttt{nv\_bf\_float162}
\begin{itemize}
\item The \texttt{nv\_bf\_float162} vector result of less-than comparison of vectors \(a\) and \(b\).
\end{itemize}
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.

```c
__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.

```c
__device__ __nv_bfloat162 __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_bfloat16` 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.

```
__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.

```
__device__ __nv_bfloat162 __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`
__device__ __nv_bfloat162 __hmax2_nan (const __nv_bfloat162 a, const __nv_bfloat162 b)
Calculates \( \text{nv}\_\text{bfloat162} \) vector maximum of two inputs, NaNs pass through.

Description
Calculates \( \text{nv}\_\text{bfloat162} \) vector \( \max[a, b] \). Elementwise \( \text{nv}\_\text{bfloat16} \) 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

__device__ __nv_bfloat162 __hmin2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
Calculates \( \text{nv}\_\text{bfloat162} \) vector minimum of two inputs.

Description
Calculates \( \text{nv}\_\text{bfloat162} \) vector \( \min[a, b] \). Elementwise \( \text{nv}\_\text{bfloat16} \) 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 \( b \)

__device__ __nv_bfloat162 __hmin2_nan (const __nv_bfloat162 a, const __nv_bfloat162 b)
Calculates \( \text{nv}\_\text{bfloat162} \) vector minimum of two inputs, NaNs pass through.

Description
Calculates \( \text{nv}\_\text{bfloat162} \) vector \( \min[a, b] \). Elementwise \( \text{nv}\_\text{bfloat16} \) 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
__device__ __nv_bfloat162 __hne2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs \texttt{nv\_bfloat162} vector not-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 not-equal comparison of vectors \texttt{a} and \texttt{b}.
\end{itemize}

Description

Performs \texttt{nv\_bfloat162} vector not-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.

__device__ unsigned __hne2\_mask (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs \texttt{nv\_bfloat162} vector not-equal comparison.

Parameters

\texttt{a}
- \texttt{nv\_bfloat162}. Is only being read.

\texttt{b}
- \texttt{nv\_bfloat162}. Is only being read.

Returns

unsigned int

\begin{itemize}
  \item The vector mask result of not-equal comparison of vectors \texttt{a} and \texttt{b}.
\end{itemize}

Description

Performs \texttt{nv\_bfloat162} vector not-equal 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 false results.
__device__ __nv_bfloat162 __hneu2 (const __nv_bfloat162 a, const __nv_bfloat162 b)

Performs `nv_bfloat162` vector unordered 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 unordered 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 true results.

__device__ unsigned __hneu2_mask (const __nv_bfloat162 a, const __nv_bfloat162 b)

Performs `nv_bfloat162` vector unordered 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 unordered 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 true results.
1.3.5. 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

Description
Converts both halves of nv_bfloat162 input a to float2 and returns the result.

__device__ __nv_bfloat162 __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

Description
Returns nv_bfloat162 number with both halves equal to the input a nv_bfloat16 number.
__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 \( \texttt{h} \) to a signed integer in round-towards-zero mode. NaN inputs are converted to 0.

\[
\textbf{\_\_device\_\_long long int \_\_bfloat162ll\_rd (const \_\_nv\_bfloat16 \texttt{h})}
\]
Convert a `nv_bfloat16` to a signed 64-bit integer in round-down mode.

Parameters

\( \texttt{h} \)
- `nv_bfloat16`. Is only being read.

Returns

`long long int`

- \( \texttt{h} \) converted to a signed 64-bit integer.

Description
Convert the `nv_bfloat16` floating-point value \( \texttt{h} \) to a signed 64-bit integer in round-down mode. NaN inputs return a `long long int` with hex value of \( 0x8000000000000000 \).

\[
\textbf{\_\_device\_\_long long int \_\_bfloat162ll\_rn (const \_\_nv\_bfloat16 \texttt{h})}
\]
Convert a `nv_bfloat16` to a signed 64-bit integer in round-to-nearest-even mode.

Parameters

\( \texttt{h} \)
- `nv_bfloat16`. Is only being read.

Returns

`long long int`

- \( \texttt{h} \) converted to a signed 64-bit integer.

Description
Convert the `nv_bfloat16` floating-point value \( \texttt{h} \) 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.
__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

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

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.
__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.

__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__ __nv_bfloat16 __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__ __nv_bfloat162 __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_bfloat162

› The nv_bfloat162 which has corresponding halves equal to the converted float2 components.

Description

Converts both components of float2 to nv_bfloat16 precision in round-to-nearest mode and combines the results into one nv_bfloat162 number. Low 16 bits of the return value correspond to a.x and high 16 bits of the return value correspond to a.y.
`__host__ __device__ __nv_bfloat16 __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.

`__host__ __device__ __nv_bfloat162 __float2bfloat162_rn (const float a)`

Converts input to `nv_bfloat16` precision in round-to-nearest-even mode and populates both halves of `nv_bfloat162` 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 `a` to `nv_bfloat16` precision in round-to-nearest-even mode and populates both halves of `nv_bfloat162` with converted value.
__host__ __device__ __nv_bfloat16 __float2bfloat16_rd (const float a)

Converts float number to __nv_bfloat16__ precision in round-down 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-down mode.

__host__ __device__ __nv_bfloat16 __float2bfloat16_rn (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.
__host____device__ __nv_bfloat16 __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__ __nv_bfloat16 __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** __nv_bfloat162 __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.

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

**Returns**

- **nv_bfloat162**
  - The nv_bfloat162 value with corresponding halves equal to the converted input floats.

**Description**

Converts both input floats to nv_bfloat16 precision in round-to-nearest-even mode and combines the results into one nv_bfloat162 number. Low 16 bits of the return value correspond to the input `a`, high 16 bits correspond to the input `b`.

**__device__** __nv_bfloat162 __halves2bfloat162 (const __nv_bfloat16 a, const __nv_bfloat16 b)

Combines two nv_bfloat16 numbers into one nv_bfloat162 number.

**Parameters**

- **a**
  - nv_bfloat16. Is only being read.

- **b**
  - nv_bfloat16. Is only being read.

**Returns**

- **nv_bfloat162**
  - The nv_bfloat162 with one nv_bfloat16 equal to `a` and the other to `b`.

**Description**

Combines two input nv_bfloat16 number `a` and `b` into one nv_bfloat162 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.
\_\_device\_\_ \_nv\_bfloa16 \_\_high2bfloa16 (const __nv\_bfloa162 a)
Returns high 16 bits of \_\_nv\_bfloa16 input.

Parameters
\textbf{a}
- \textit{nv\_bfloa16}. Is only being read.

Returns
\textit{nv\_bfloa16}
\begin{itemize}
\item The high 16 bits of the input.
\end{itemize}

Description
Returns high 16 bits of \_\_nv\_bfloa16 input \textit{a}.

\_\_device\_\_ \_nv\_bfloa162 \_\_high2bfloa162 (const __nv\_bfloa162 a)
Extracts high 16 bits from \_\_nv\_bfloa162 input.

Parameters
\textbf{a}
- \textit{nv\_bfloa162}. Is only being read.

Returns
\textit{nv\_bfloa162}
\begin{itemize}
\item The \textit{nv\_bfloa162} with both halves equal to the high 16 bits of the input.
\end{itemize}

Description
Extracts high 16 bits from \_\_nv\_bfloa162 input \textit{a} and returns a new \textit{nv\_bfloa162} 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

\textbf{a}
- __nv_bfloat162. Is only being read.

Returns

float
- The high 16 bits of \textit{a} converted to float.

Description

Converts high 16 bits of __nv_bfloat162 input \textit{a} to 32-bit floating-point number and returns the result.

__device__ __nv_bfloat162 __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

\textbf{a}
- __nv_bfloat162. Is only being read.

\textbf{b}
- __nv_bfloat162. Is only being read.

Returns

__nv_bfloat162
- The high 16 bits of \textit{a} and of \textit{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 \textit{a} is stored in low 16 bits of the return value, high 16 bits from input \textit{b} is stored in high 16 bits of the return value.
__device__ __nv_bfloat16 __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__ __nv_bfloat16 __int2bfloat16_rn (const int i)
Convert a signed integer to a nv_bfloat16 in round-to-nearest-even 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-to-nearest-even mode.

__device__ __nv_bfloat16 __int2bfloat16_ru (const int i)
Convert a signed integer to a nv_bfloat16 in round-up mode.

Parameters
i
  - int. Is only being read.
## __device__ __nv_bfloat16 __int2bfloat16_rz (const int i)

Convert a signed integer to a nv_bfloat16 in round-towards-zero mode.

**Parameters**

- `i` - int. Is only being read.

**Returns**

- `nv_bfloat16` converted to `nv_bfloat16`.

## __device__ __nv_bfloat16 __ldca (const __nv_bfloat16 *ptr)

Generates a `ld.global.ca` load instruction.

**Parameters**

- `ptr` - memory location

**Returns**

The value pointed by `ptr`
__device__ __nv_bfloat16 __ldca (const __nv_bfloat16 *ptr)
Generates a `ld.global.ca` load instruction.

Parameters
ptr
- memory location

Returns
The value pointed by `ptr`

__device__ __nv_bfloat16 __ldcg (const __nv_bfloat16 *ptr)
Generates a `ld.global.cg` load instruction.

Parameters
ptr
- memory location

Returns
The value pointed by `ptr`

__device__ __nv_bfloat16 __ldcg (const __nv_bfloat16 *ptr)
Generates a `ld.global.cg` load instruction.

Parameters
ptr
- memory location

Returns
The value pointed by `ptr`

__device__ __nv_bfloat16 __ldcs (const __nv_bfloat16 *ptr)
Generates a `ld.global.cs` load instruction.

Parameters
ptr
- memory location
Returns
The value pointed by `ptr`

```c
__device__ __nv_bfloat16 __ldcs (const __nv_bfloat16 *ptr)
```
Generates a `ld.global.cs` load instruction.

Parameters
- **ptr** - memory location

Returns
The value pointed by `ptr`

```c
__device__ __nv_bfloat16 __ldcv (const __nv_bfloat16 *ptr)
```
Generates a `ld.global.cv` load instruction.

Parameters
- **ptr** - memory location

Returns
The value pointed by `ptr`
__device__ __nv_bfloat16 __ldg (const __nv_bfloat16 *ptr)
Generates a `ld.global.nc` load instruction.

Parameters
ptr
- memory location

Returns
The value pointed by `ptr`

__device__ __nv_bfloat162 __ldg (const __nv_bfloat162 *ptr)
Generates a `ld.global.nc` load instruction.

Parameters
ptr
- memory location

Returns
The value pointed by `ptr`

__device__ __nv_bfloat16 __ldlu (const __nv_bfloat16 *ptr)
Generates a `ld.global.lu` load instruction.

Parameters
ptr
- memory location

Returns
The value pointed by `ptr`

__device__ __nv_bfloat162 __ldlu (const __nv_bfloat162 *ptr)
Generates a `ld.global.lu` load instruction.

Parameters
ptr
- memory location

Returns
The value pointed by `ptr`
__device__ __nv_bfloat16 __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__ __nv_bfloat16 __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__ __nv_bfloat16 __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__ __nv_bfloat16 __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.
__device__ __nv_bfloat16 __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 `nv_bfloat16` which contains low 16 bits of the input `a`.

Description
Returns low 16 bits of `nv_bfloat162` input `a`.

__device__ __nv_bfloat162 __low2bfloat162 (const __nv_bfloat162 a)
Extracts low 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 low 16 bits of the input.

Description
Extracts low 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 __low2float (const __nv_bfloat162 a)\**

Converts low 16 bits of nv_bfloat162 to float and returns the result.

**Parameters**

\texttt{a}

- nv_bfloat162. Is only being read.

**Returns**

float

\>

The low 16 bits of \texttt{a} converted to float.

**Description**

Converts low 16 bits of nv_bfloat162 input \texttt{a} to 32-bit floating-point number and returns the result.

**__device__ __nv_bfloat162 __lowhigh2highlow (const __nv_bfloat162 a)\**

Swaps both halves of the nv_bfloat162 input.

**Parameters**

\texttt{a}

- nv_bfloat162. Is only being read.

**Returns**

nv_bfloat162

\>

\texttt{a} with its halves being swapped.

**Description**

Swaps both halves of the nv_bfloat162 input and returns a new nv_bfloat162 number with swapped halves.
__device__ __nv_bfloat162 __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**

\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**

\texttt{nv\_bfloat162}

\begin{itemize}
  \item The low 16 bits of \texttt{a} and of \texttt{b}.
\end{itemize}

**Description**

Extracts low 16 bits from each of the two \texttt{nv\_bfloat162} inputs and combines into one \texttt{nv\_bfloat162} 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__ __nv_bfloat16 __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**

\begin{itemize}
  \item \texttt{mask} - unsigned int. Is only being read.
  \item \texttt{var} - \texttt{nv\_bfloat16}. Is only being read.
  \item \texttt{delta} - int. Is only being read.
  \item \texttt{width} - int. Is only being read.
\end{itemize}
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__ __nv_bfloat162 __shfl_down_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 higher 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 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.

```c
__device__ __nv_bfloat16 __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.
- **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**

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
Exchange a variable between threads within a warp. Direct copy from indexed thread.

**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**

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__ __nv_bfloat16 __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
- 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 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__ __nv_bfloat162 __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__ __nv_bfloat16 __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__ __nv_bfloat16 __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 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 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__ __nv_bfloat16 __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__ __nv_bfloat16 __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__ __nv_bfloat16 __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__ __nv_bfloat16 __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.
__device__ __nv_bfloat16 __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__ __nv_bfloat16 __uint2bfloat16_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.
**__host__** __device__ **__nv_bfloat16** **__uint2bf10at16__** __rn (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.

**__device__** **__nv_bfloat16** **__uint2bf10at16__** __ru (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__ __nv_bfloat16 __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__ __nv_bfloat16 __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__ __nv_bfloat16 __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__ __nv_bfloat16 __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__ __nv_bfloat16 __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__ __nv_bfloat16 __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__ __nv_bfloat16 __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__ __nv_bfloat16 __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__ __nv_bfloat16 __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.

__device__ __nv_bfloat16 __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.

1.3.6. Bfloat16 Math Functions

Bfloat16 Precision Intrinsics

To use these functions, include the header file cuda_bf16.h in your program.
__device__ __nv_bfloat16 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__ __nv_bfloat16 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.

__device__ __nv_bfloat16 hexp (const __nv_bfloat16 a)
Calculates nv_bfloat16 natural exponential function in round-to-nearest mode.

Parameters
a
- nv_bfloat16. Is only being read.

Returns
nv_bfloat16
- The natural exponential function on a.
Description
Calculates \texttt{nv\_bfloat16} natural exponential function of input \(a\) in round-to-nearest-even mode.

\texttt{\_\_device\_\_nv\_bfloat16\ hexp10 (const\_\_nv\_bfloat16\ a)}
Calculates \texttt{nv\_bfloat16} decimal exponential function in round-to-nearest mode.

Parameters
\(a\)
- \texttt{nv\_bfloat16}. Is only being read.

Returns
\texttt{nv\_bfloat16}
- The decimal exponential function on \(a\).

Description
Calculates \texttt{nv\_bfloat16} decimal exponential function of input \(a\) in round-to-nearest-even mode.

\texttt{\_\_device\_\_nv\_bfloat16\ hexp2 (const\_\_nv\_bfloat16\ a)}
Calculates \texttt{nv\_bfloat16} binary exponential function in round-to-nearest mode.

Parameters
\(a\)
- \texttt{nv\_bfloat16}. Is only being read.

Returns
\texttt{nv\_bfloat16}
- The binary exponential function on \(a\).

Description
Calculates \texttt{nv\_bfloat16} binary exponential function of input \(a\) in round-to-nearest-even mode.
__device__ __nv_bfloat16 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__ __nv_bfloat16 hlog (const __nv_bfloat16 a)
Calculates nv_bfloat16 natural logarithm in round-to-nearest-even mode.

Parameters
a
- nv_bfloat16. Is only being read.

Returns
nv_bfloat16
- The natural logarithm of a.

Description
Calculates nv_bfloat16 natural logarithm of input a in round-to-nearest-even mode.

__device__ __nv_bfloat16 hlog10 (const __nv_bfloat16 a)
Calculates nv_bfloat16 decimal logarithm in round-to-nearest-even mode.

Parameters
a
- nv_bfloat16. Is only being read.

Returns
nv_bfloat16
- The decimal logarithm of a.
**Description**
Calculates \texttt{nv\_bf\_a\it{}log16} decimal logarithm of input \texttt{a} in round-to-nearest-even mode.

```c
__device__ __nv\_bf\_a\it{}log16 hlog2 (const __nv\_bf\_a\it{} a)
```
Calculates \texttt{nv\_bf\_a\it{}log16} binary logarithm in round-to-nearest-even mode.

**Parameters**
\texttt{a}
- \texttt{nv\_bf\_a\it{}}. Is only being read.

**Returns**
\texttt{nv\_bf\_a\it{}}
- The binary logarithm of \texttt{a}.

**Description**
Calculates \texttt{nv\_bf\_a\it{}log16} binary logarithm of input \texttt{a} in round-to-nearest-even mode.

```c
__device__ __nv\_bf\_a\it{}log16 hrcp (const __nv\_bf\_a\it{} a)
```
Calculates \texttt{nv\_bf\_a\it{}log16} reciprocal in round-to-nearest-even mode.

**Parameters**
\texttt{a}
- \texttt{nv\_bf\_a\it{}}. Is only being read.

**Returns**
\texttt{nv\_bf\_a\it{}}
- The reciprocal of \texttt{a}.

**Description**
Calculates \texttt{nv\_bf\_a\it{}log16} reciprocal of input \texttt{a} in round-to-nearest-even mode.

```c
__device__ __nv\_bf\_a\it{}log16 hrint (const __nv\_bf\_a\it{} h)
```
Round input to nearest integer value in \texttt{nv\_bf\_a\it{}} floating-point number.

**Parameters**
\texttt{h}
- \texttt{nv\_bf\_a\it{}}. Is only being read.
Returns

`nv_bfloat16`

- The nearest integer to `h`.

Description

Round `h` to the nearest integer value in `nv_bfloat16` floating-point format, with bfloat16way cases rounded to the nearest even integer value.

```c
__device__ __nv_bfloat16 hrsqrt (const __nv_bfloat16 a)
```

Calculates `nv_bfloat16` reciprocal square root in round-to-nearest-even mode.

Parameters

`a`
- `nv_bfloat16`. Is only being read.

Returns

`nv_bfloat16`

- The reciprocal square root of `a`.

Description

Calculates `nv_bfloat16` reciprocal square root of input `a` in round-to-nearest mode.

```c
__device__ __nv_bfloat16 hsin (const __nv_bfloat16 a)
```

Calculates `nv_bfloat16` sine in round-to-nearest-even mode.

Parameters

`a`
- `nv_bfloat16`. Is only being read.

Returns

`nv_bfloat16`

- The sine of `a`.

Description

Calculates `nv_bfloat16` sine of input `a` in round-to-nearest-even mode.
__device__ __nv_bfloat16 hsqrt (const __nv_bfloat16 a)
Calculates nv_bfloat16 square root in round-to-nearest-even mode.

Parameters
a  
- nv_bfloat16. Is only being read.

Returns
nv_bfloat16
- The square root of a.

Description
Calculates nv_bfloat16 square root of input a in round-to-nearest-even mode.

__device__ __nv_bfloat16 htrunc (const __nv_bfloat16 h)
Truncate input argument to the integral part.

Parameters
h  
- 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.

1.3.7. Bfloat162 Math Functions
Bfloat16 Precision Intrinsics
To use these functions, include the header file cuda_bf16.h in your program.
__device__ __nv_bfloat162 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
\( \text{nv\_bfloat162} \)
- The vector of smallest integers not less than \( h \).

Description
For each component of vector \( h \) compute the smallest integer value not less than \( h \).

__device__ __nv_bfloat162 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.

__device__ __nv_bfloat162 h2exp (const __nv_bfloat162 a)
Calculates \( \text{nv\_bfloat162} \) vector exponential function in round-to-nearest mode.

Parameters
\( a \)
- \( \text{nv\_bfloat162} \). Is only being read.

Returns
\( \text{nv\_bfloat162} \)
- The elementwise exponential function on vector \( a \).
Description
Calculates `nv_bfloat162` exponential function of input vector `a` in round-to-nearest-even mode.

```__device__ __nv_bfloat162 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__ __nv_bfloat162 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__ __nv_bfloat162 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__ __nv_bfloat162 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__ __nv_bfloat162 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 \texttt{nv\_bfloat162} decimal logarithm of input vector a in round-to-nearest-even mode.

\texttt{__device\_ __nv\_bfloat162 h2log2 (const __nv\_bfloat162 a)}
Calculates \texttt{nv\_bfloat162} vector binary logarithm in round-to-nearest-even mode.

Parameters
a
- \texttt{nv\_bfloat162}. Is only being read.

Returns
nv_bfloat162
➤ The elementwise binary logarithm on vector a.

Description
Calculates \texttt{nv\_bfloat162} binary logarithm of input vector a in round-to-nearest mode.

\texttt{__device\_ __nv\_bfloat162 h2rcp (const __nv\_bfloat162 a)}
Calculates \texttt{nv\_bfloat162} vector reciprocal in round-to-nearest-even mode.

Parameters
a
- \texttt{nv\_bfloat162}. Is only being read.

Returns
nv_bfloat162
➤ The elementwise reciprocal on vector a.

Description
Calculates \texttt{nv\_bfloat162} reciprocal of input vector a in round-to-nearest-even mode.
__device__ __nv_bfloat162 h2rint (const __nv_bfloat162 h)
Round input to nearest integer value in nv_bfloat16 floating-point number.

Parameters
h
- nv_bfloat162. Is only being read.

Returns
nv_bfloat162
▶ The vector of rounded integer values.

Description
Round each component of nv_bfloat162 vector h to the nearest integer value in nv_bfloat16 floating-point format, with bfloat16way cases rounded to the nearest even integer value.

__device__ __nv_bfloat162 h2rsqrt (const __nv_bfloat162 a)
Calculates nv_bfloat162 vector reciprocal square root in round-to-nearest mode.

Parameters
a
- nv_bfloat162. Is only being read.

Returns
nv_bfloat162
▶ The elementwise reciprocal square root on vector a.

Description
Calculates nv_bfloat162 reciprocal square root of input vector a in round-to-nearest-even mode.

__device__ __nv_bfloat162 h2sin (const __nv_bfloat162 a)
Calculates nv_bfloat162 vector sine in round-to-nearest-even mode.

Parameters
a
- nv_bfloat162. Is only being read.
Returns
nv_bfloat162
▶ The elementwise sine on vector a.

Description
Calculates nv_bfloat162 sine of input vector a in round-to-nearest-even mode.

__device__ __nv_bfloat162 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 mode.

__device__ __nv_bfloat162 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:

```c
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.

```c
__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 \([-1, +1]\).

- \(\text{acosf}(1)\) returns +0.
- \(\text{acosf}(x)\) returns NaN for \(x\) outside \([-1, +1]\).

Description

Calculate the principal value of the arc cosine of the input argument \(x\).
**__device__ float acoshf (float x)**

Calculate the nonnegative inverse hyperbolic cosine of the input argument.

**Returns**

Result will be in the interval \([0, +\infty]\).

- \(\text{acoshf}(1)\) returns 0.
- \(\text{acoshf}(x)\) returns \(\text{NaN}\) for \(x\) in the interval \([-\infty, 1]\).
- \(\text{acoshf}(+\infty)\) returns \(+\infty\).

**Description**

Calculate the nonnegative inverse hyperbolic cosine of the input argument \(x\).

---

**__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 \(\text{NaN}\) for \(x\) outside \([-1, +1]\).

**Description**

Calculate the principal value of the arc sine of the input argument \(x\).
__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 \([- \pi, + \pi]\).

- \( \text{atan2f}(\pm 0, -0) \) returns \( \pm \pi \).
- \( \text{atan2f}(\pm 0, +0) \) returns \( \pm 0 \).
- \( \text{atan2f}(\pm 0, x) \) returns \( \pm \pi \) for \( x < 0 \).
- \( \text{atan2f}(\pm 0, x) \) returns \( \pm 0 \) for \( x > 0 \).
- \( \text{atan2f}(y, \pm 0) \) returns \( -\pi/2 \) for \( y < 0 \).
- \( \text{atan2f}(y, \pm 0) \) returns \( \pi/2 \) for \( y > 0 \).
- \( \text{atan2f}(\pm y, -\infty) \) returns \( \pm \pi \) for finite \( y > 0 \).
- \( \text{atan2f}(\pm y, +\infty) \) returns \( \pm 0 \) for finite \( y > 0 \).
- \( \text{atan2f}(\pm \infty, x) \) returns \( \pm \pi/2 \) for finite \( x \).
- \( \text{atan2f}(\pm \infty, -\infty) \) returns \( \pm 3\pi/4 \).
- \( \text{atan2f}(\pm \infty, +\infty) \) returns \( \pm \pi/4 \).
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.

`__device__ float atanf(float x)`
Calculate the arc tangent of the input argument.

Returns
Result will be in radians, in the interval \([-\pi/2, +\pi/2]\).

- `atanf(\pm 0)` returns \( \pm 0 \).
- `atanf(\pm \infty)` returns \( \pm \pi/2 \).

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, Single-Precision Floating-Point Functions section.

`__device__ float atanhf(float x)`
Calculate the inverse hyperbolic tangent of the input argument.

Returns
- `atanhf(\pm 0)` returns \( \pm 0 \).
- `atanhf(\pm 1)` returns \( \pm \infty \).
- `atanhf(x)` returns NaN for \( x \) outside interval \([-1, 1]\).

Description
Calculate the inverse hyperbolic tangent of the input argument \( x \).
__device__ float cbrtf (float x)
Calculate the cube root of the input argument.

Returns
Returns \( \sqrt[3]{x} \).

- \( \text{cbrtf}(\pm 0) \) returns \( \pm 0 \).
- \( \text{cbrtf}(\pm \infty) \) returns \( \pm \infty \).

Description
Calculate the cube root of \( x \), \( \sqrt[3]{x} \).

__device__ float ceilf (float x)
Calculate ceiling of the input argument.

Returns
Returns \( \lfloor x \rfloor \) expressed as a floating-point number.

- \( \text{ceilf}(\pm 0) \) returns \( \pm 0 \).
- \( \text{ceilf}(\pm \infty) \) 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.
__device__ float cospif (float x)
Calculate the cosine of the input argument \( x \times \pi \).

Returns
- \( \cos(0) \) returns 1.
- \( \cos(\pm \infty) \) returns NaN.

Description
Calculate the cosine of \( x \times \pi \) [measured in radians], where \( x \) is the input argument.

__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 \( x \), \( I_0(x) \).
__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 \( -\infty \).
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].

- **Note:**
  For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float erfcxf(float x)

Calculate the scaled complementary error function of the input argument.

Returns

- erfcxf($-\infty$) returns $+\infty$.
- erfcxf($+\infty$) returns $0$.

Description

Calculate the scaled complementary error function of the input argument $x$, $e^{x^2} \cdot \text{erfc}(x)$.

- **Note:**
  For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float erff(float x)

Calculate the error function of the input argument.

Returns

- erff($\pm 0$) returns $\pm 0$.
- erff($\pm \infty$) returns $1$.

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].

__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 \( x \).


**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**

- `exp2f(±0)` returns 1.
- `exp2f(−∞)` returns +0.
- `exp2f(+∞)` 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, Single-Precision Floating-Point Functions section.

---

### __device__ float expf (float x)

Calculate the base $e$ exponential of the input argument.

**Returns**

- `expf(±0)` returns 1.
- `expf(−∞)` returns +0.
- `expf(+∞)` returns +∞.

**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 $+\infty$.

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 $+\infty$.
- `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 \(y\).

- \(\text{fdimf}(x, y)\) returns \(x - y\) if \(x > y\).
- \(\text{fdimf}(x, y)\) 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\((\pm\infty)\) returns \(\pm\infty\).
- floorf\((\pm0)\) returns \(\pm0\).

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 \(x \times y + z\) as a single operation.

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
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, 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.
- fmodf(±0, y) returns ±0 if y is not zero.
- fmodf(x, ±∞) returns x if x is finite.
- fmodf(x, y) returns NaN if x is ±∞ 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 frexpf (float x, int *nptr)
Extract mantissa and exponent of a floating-point value.

Returns

Returns the fractional component m.

- frexpf(±0, nptr) returns ±0 and stores zero in the location pointed to by nptr.
- frexpf(±∞, nptr) returns ±∞ and stores an unspecified value in the location to which nptr points.
- frexpf(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 \( \text{nptr} \) points.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__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 \( \sqrt{x^2 + y^2} \).

- hypotf\( x, y \), hypotf\( y, x \), and hypotf\( x, -y \) are equivalent.
- hypotf\( x, \pm 0 \) is equivalent to fabsf\( x \).
- 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.

__device__ int ilogbf (float x)
Compute the unbiased integer exponent of the argument.

Returns
- If successful, returns the unbiased exponent of the argument.
- ilogbf\( \pm 0 \) returns INT_MIN.
- ilogbf(NaN) returns INT_MIN.
- ilogbf\( \pm \infty \) returns 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, Single-Precision Floating-Point Functions section.

__device__ __RETURN_TYPE isnan (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.

- j0f(±∞) returns +0.
- j0f(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.

- j1f(±0) returns ±0.
- j1f(±∞) returns ±0.
Modules

CUDA Math API

vRelease Version | 242

- 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.

```c
__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.

```c
__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 \( \exp \).
Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__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 \[ \log_e \left( \int_0^\infty e^{-t} t^{x-1} dt \right) \]

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__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 \( \	ext{lrintf} \). 

__device__ float log10f (float x)
Calculate the base 10 logarithm of the input argument.

Returns
- \( \log_{10}f(\pm 0) \) returns \(-\infty\).
- \( \log_{10}f(1) \) returns \(+0\).
- \( \log_{10}f(x) \) returns NaN for \( x < 0 \).
- \( \log_{10}f(\pm \infty) \) 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 \(--\text{use\_fast\_math} \) 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 $x < -1$.
- $\log1pf(\pm \infty)$ returns $\pm \infty$.

**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 $x < 0$.
- $\log2f(\pm \infty)$ returns $\pm \infty$.

**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.
__device__ float logbf (float x)
Calculate the floating-point representation of the exponent of the input argument.

Returns
- logbf(±0) returns −∞.
- logbf(±∞) returns +∞.

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 −∞.
- logf(1) returns +0.
- logf(x) returns NaN for x < 0.
- logf(+∞) returns +∞.

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 \( \text{lrintf} \).  

__device__ float max (const float a, const float b)
Calculate the maximum value of the input \( \text{float} \) arguments.

Description
Calculate the maximum value of the arguments \( a \) and \( b \). Behavior is equivalent to \( \text{fmaxf} \) function.

Note, this is different from \( \text{std}:: \) 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.
```

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
__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, Φ(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, Φ^−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 \(\pm\, 0\) for \(y\) an odd integer greater than 0.
- \(\text{powf}(\pm\, 0, \, y)\) returns +0 for \(y > 0\) and not an odd integer.
- \(\text{powf}(-1, \, y)\) 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 +0 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.
Module: CUDA Math API

Release Version: 253

- `powf(-\infty, y)` returns $0$ for $y < 0$ and not an odd integer.
- `powf(-\infty, y)` returns $-\infty$ for $y$ an odd integer greater than 0.
- `powf(-\infty, y)` returns $+\infty$ for $y > 0$ and not an odd integer.
- `powf(+\infty, y)` returns $0$ for $y < 0$.
- `powf(+\infty, y)` returns $+\infty$ 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.

```c
__device__ float rcbrtf(float x)
```

Calculate reciprocal cube root function.

**Returns**
- `rcbrt(\pm 0)` returns $\pm \infty$.
- `rcbrt(\pm \infty)` returns $\pm 0$.

**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

Compute 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

Compute 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.

__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 \( \frac{1}{\sqrt{x^2 + y^2}} \).

- \( \text{rhypotf}(x, y) \), \( \text{rhypotf}(y, x) \), and \( \text{rhypotf}(x, -y) \) are equivalent.
- \( \text{rhypotf}(\pm \infty, y) \) 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.

__device__ float rintf (float x)
Round input to nearest integer value in floating-point.

Returns
Returns rounded integer value.

- \( \text{rintf}(\pm 0) \) returns \( \pm 0 \).
- \( \text{rintf}(\pm \infty) \) 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.
__device__ float rnorm3df (float a, float b, float 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 $+\infty$ is returned, even if there are NaNs.

Description
Calculates one over the length of three dimension 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 rnorm4df (float a, float b, float c, float 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 $+\infty$ is returned, even if there are NaNs.

Description
Calculates one over the length of four dimension 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 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 +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.

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(±0) returns ±0.
‣ roundf(±∞) 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\_\_ float rsqrtf (float x)
Calculate the reciprocal of the square root of the input argument.

**Returns**

Returns \(1/\sqrt{x}\).
- \(\text{rsqrtf}(+\infty)\) returns +0.
- \(\text{rsqrtf}(\pm 0)\) returns \(\pm\infty\).
- \(\text{rsqrtf}(x)\) 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 \(x \times 2^n\).
- \(\text{scalblnf}(\pm 0, n)\) returns \(\pm 0\).
- \(\text{scalblnf}(x, 0)\) returns \(x\).
- \(\text{scalblnf}(\pm\infty, n)\) 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.

```c
__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.

```c
__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().
__device__ void sincospif (float x, float *sptr, float *cptr)
Calculate the sine and cosine of the first input argument \( x \times \pi \).

Returns
none

Description
Calculate the sine and cosine of the first input argument, \( x \) (measured in radians), \( x \times \pi \). The results for sine and cosine are written into the second argument, \( \text{sptr} \), and, respectively, third argument, \( \text{cptr} \).

See also:
sinpif() and 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
- sinf( ± 0 ) returns ± 0.
- sinf( ± \infty ) 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(±0) returns ±0.
- sinhf(±∞) returns ±∞.

Description
Calculate the hyperbolic sine of the input argument x.

__device__ float sinpif (float x)
Calculate the sine of the input argument \( x \pi \).

Returns
- sinpif(±0) returns ±0.
- sinpif(±∞) returns NaN.

Description
Calculate the sine 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 sqrtf(float x)
Calculate the square root of the input argument.

Returns
Returns $\sqrt{x}$:
- $\sqrt{0}$ returns $0$.
- $\sqrt{\infty}$ 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, Single-Precision Floating-Point Functions section.

__device__ float tanf(float x)
Calculate the tangent of the input argument.

Returns
- $\tan(0)$ returns $0$.
- $\tan(\infty)$ 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, 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 tanhf (float x)
Calculate the hyperbolic tangent of the input argument.

Returns
- \( \tanhf(\pm 0) \) returns \( \pm 0 \).
- \( \tanhf(\pm \infty) \) 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
- \( \tgammaf(\pm 0) \) returns \( \pm \infty \).
- \( \tgammaf(2) \) returns +1.
- \( \tgammaf(x) \) returns NaN if \( x < 0 \) and \( x \) is an integer.
- \( \tgammaf(-\infty) \) returns NaN.
- \( \tgammaf(+\infty) \) 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.

- `truncf(±0)` returns ±0.
- `truncf(±∞)` returns ±∞.

**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.

- `y0f(±0)` returns −∞.
- `y0f(x)` returns NaN for `x < 0`.
- `y0f(±∞)` returns +0.
- `y0f(NaN)` returns NaN.

**Description**
Calculate the value of the Bessel function of the second kind of order 0 for the input argument x, \( Y_0(x) \).

**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.
• y1f(±0) returns -∞.
• y1f(x) returns NaN for x < 0.
• y1f(∞) returns +0.
• y1f(NaN) returns NaN.

Description
Calculate the value of the Bessel function of the second kind of order 1 for the input argument x, \( Y_1(x) \).

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float ynf (int n, float 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.
• ynf(n, x) returns NaN for n < 0.
• ynf(n, ±0) returns -∞.
• ynf(n, x) returns NaN for x < 0.
• ynf(n, ∞) returns +0.
• ynf(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, 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 \(x\).
__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 \(x\).

__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 [-\pi/2, +\pi/2].

- \text{atan}(±0) returns ±0.
- \text{atan}(±\infty) returns ±\pi/2.

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, 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 [-\pi, +\pi].

- \text{atan2}(±0, -0) returns ±\pi.
- \text{atan2}(±0, +0) returns ±0.
- \text{atan2}(±0, x) returns ±\pi for x < 0.
- \text{atan2}(±0, x) returns ±0 for x > 0.
- \text{atan2}(y, ±0) returns -\pi/2 for y < 0.
- \text{atan2}(y, ±0) returns \pi/2 for y > 0.
- \text{atan2}(±y, -\infty) returns ±\pi for finite y > 0.
- \text{atan2}(±y, +\infty) returns ±0 for finite y > 0.
- \text{atan2}(±\infty, x) returns ±\pi/2 for finite x.
- \text{atan2}(±\infty, -\infty) returns ±3\pi/4.
- \text{atan2}(±\infty, +\infty) 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, 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.

- $\text{ceil}(\pm 0)$ returns $\pm 0$.
- $\text{ceil}(\pm \infty)$ 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

- $\text{cos}(\pm 0)$ returns 1.
- $\text{cos}(\pm \infty)$ 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, Double-Precision Floating-Point Functions section.

__device__ double cosh (double x)
Calculate the hyperbolic cosine of the input argument.

Returns
- \( \cosh( \pm 0) \) returns 1.
- \( \cosh( \pm \infty) \) returns \( +\infty \).

Description
Calculate the 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 cospi (double x)
Calculate the cosine of the input argument \( x \times \pi \).

Returns
- \( \cospi( \pm 0) \) returns 1.
- \( \cospi( \pm \infty) \) returns NaN.

Description
Calculate the cosine 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 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 \( x \), 
\[
\frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt.
\]

**Note:**
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

**__device__ double erfc (double x)**

Calculate the complementary error function of the input argument.

**Returns**

- \( \text{erfc}( -\infty ) \) returns 2.
- \( \text{erfc}( +\infty ) \) 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, Double-Precision Floating-Point Functions section.

**__device__ double erfcinv (double x)**

Calculate the inverse complementary error function of the input argument.

**Returns**

- \( \text{erfcinv}( \pm 0 ) \) returns +\( \infty \).
- \( \text{erfcinv}(2) \) returns \(- \infty \).
- \( \text{erfcinv}(x) \) 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__ double erfcx (double x)

Calculate the scaled complementary error function of the input argument.

Returns

- \( \text{erfcx}(\infty) \) returns +\( \infty \).
- \( \text{erfcx}(0) \) 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

- \( \text{erfinv}(\pm 0) \) returns \( \pm 0 \).
- \( \text{erfinv}(1) \) returns +\( \infty \).
- \( \text{erfinv}(-1) \) returns -\( \infty \).
- \( \text{erfinv}(x) \) 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
- $\exp(\pm 0)$ 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, Double-Precision Floating-Point Functions section.

__device__ double exp10 (double x)
Calculate the base 10 exponential of the input argument.

Returns
- $\exp10(\pm 0)$ returns 1.
- $\exp10(-\infty)$ returns +0.
- $\exp10(+\infty)$ 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.

- \( \text{fabs}(\pm \infty) \) returns \( +\infty \).
- \( \text{fabs}(\pm 0) \) 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 \( y \).

- \( \text{fdim}(x, y) \) returns \( x - y \) if \( x > y \).
- \( \text{fdim}(x, y) \) 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__ 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 \( x / y \).
- \( \text{fmod}(\pm 0, y) \) returns \( \pm 0 \) if \( y \) is not zero.
- \( \text{fmod}(x, \pm \infty) \) returns \( x \) if \( x \) is finite.
- \( \text{fmod}(x, y) \) 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 \( m \).
- \( \text{frexp}(\pm 0, \text{nptr}) \) returns \( \pm 0 \) and stores zero in the location pointed to by \( \text{nptr} \).
- \( \text{frexp}(\pm \infty, \text{nptr}) \) returns \( \pm \infty \) and stores an unspecified value in the location to which \( \text{nptr} \) points.
- \( \text{frexp}(\text{NaN}, y) \) returns a NaN and stores an unspecified value in the location to which \( \text{nptr} \) 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 \(nptr\) points.

**Note:**
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

```c
__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 \(\sqrt{x^2 + y^2}\).

- \(\text{hypot}(x,y), \text{hypot}(y,x), \text{and hypot}(x,-y)\) are equivalent.
- \(\text{hypot}(x, \pm 0)\) is equivalent to \(\text{fabs}(x)\).
- \(\text{hypot}(\pm \infty, y)\) 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.

```c
__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 \(\text{FP_ILOGB0}\) 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, 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 isinf (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( ±∞ ) 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( ±0 ) returns ±0.
- j1( ±∞ ) returns ±0.
• j1(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.

```__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\( n, NaN \) 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.

```__device__ double ldexp (double x, int exp)```
Calculate the value of \( x \cdot 2^{\text{exp}} \).

Returns
• ldexp\( x, \text{exp} \) is equivalent to scalbn\( x, \text{exp} \).

Description
Calculate the value of \( x \cdot 2^{\text{exp}} \) of the input arguments \( x \) and \( \text{exp} \).
__device__ double lgamma (double x)
Calculate the natural logarithm of the absolute value of the gamma function of the input argument.

Returns
- lgamma(1) returns +0.
- lgamma(2) returns +0.
- lgamma(x) returns +∞ if x ≤ 0 and x is an integer.
- lgamma(−∞) returns +∞.
- lgamma(+∞) returns +∞.

Description
Calculate the natural logarithm of the absolute value of the gamma function of the input argument x, namely the value of \( \log_e \int_0^\infty e^{-t}t^{x-1}dt \).

__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().

__device__ double log (double x)
Calculate the base e logarithm of the input argument.

Returns
- log( ±0 ) returns -∞.
- log(1) returns +0.
- log(x) returns NaN for x < 0.
- log( +∞ ) returns +∞.

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( ±0 ) returns -∞.
- \( \log_{10}(1) \) returns +0.
- \( \log_{10}(x) \) returns \( \text{NaN} \) 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.

__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 \( \text{NaN} \) 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.

__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**

- \( \logb( \pm 0 ) \) returns \( -\infty \).
- \( \logb( \pm \infty ) \) 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 float 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(±x, iptr) returns a result with the same sign as x.
- 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 \( \sqrt{p_0^2 + p_1^2 + ... + p_{dim-1}^2} \).

- 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

- `normcdf( +∞ )` returns 1.
- `normcdf( −∞ )` returns +0.

Description

Calculate the cumulative distribution function of the standard normal distribution for input argument `x`, `Φ(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

- `normcdfinv( ±0 )` returns `−∞`.
- `normcdfinv(1)` returns `+∞`.
- `normcdfinv(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`, `Φ⁻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**

- `rcbrt(±0)` returns ±∞.
- `rcbrt(±∞)` 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 `x/y`. In the case when `|n - x/y| = 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 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.

**Note:**
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

### __device__ double rint (double x)
Round to nearest integer value in floating-point.

**Returns**
Returns rounded integer value.
- \( \text{rint}( \pm 0 ) \) returns \( \pm 0 \).
- \( \text{rint}( \pm \infty ) \) 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.

### __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 \( \pm 0 \) 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.

**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( +\infty )` 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 \( x \cdot 2^n \).
- \( \text{scalbln}(\pm 0, n) \) 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 \( x \cdot 2^n \).
- \( \text{scalbn}(\pm 0, n) \) 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.

\textbf{__device__ double sin (double x)}
Calculate the sine of the input argument.

Returns

- \( \sin(\pm0) \) returns \( \pm0 \).
- \( \sin(\pm\infty) \) returns NaN.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

\textbf{__device__ void sincos (double x, double *sptr, double *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, \( \text{sptr} \), and, respectively, third argument, \( \text{cptr} \).

See also:
\texttt{sin()} and \texttt{cos()}.

Note:
__device__ void sincospi (double x, double *sptr, double *cptr)
Calculate the sine and cosine of the first input argument \( \times \pi \).

Returns

- none

Description

Calculate the sine and cosine of the first input argument, \( x \) (measured in radians), \( \times \pi \). 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().

__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
- \( \sinpi(\pm 0) \) returns \( \pm 0 \).
- \( \sinpi(\pm \infty) \) 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
- \( \sqrt{x} \).
- \( \sqrt{\pm 0} \) 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.
- \( \text{tgamma}(x) \) returns NaN if \( x < 0 \) and \( x \) is an integer.
- \( \text{tgamma}(-\infty) \) returns NaN.
- \( \text{tgamma}(+\infty) \) returns +\( \infty \).

**Description**

Calculate the gamma function of the input argument \( x \), namely the value of \( \int_0^\infty e^{-t^x}dt \).

**Note:**

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

---

`__device__ double trunc (double x)`

Truncate input argument to the integral part.

**Returns**

Returns truncated integer value.

- \( \text{trunc}(\pm 0) \) returns \( \pm 0 \).
- \( \text{trunc}(\pm \infty) \) returns \( \pm \infty \).

**Description**

Round \( x \) to the nearest integer value that does not exceed \( x \) in magnitude.

---

`__device__ double y0 (double 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.

- \( y0(\pm 0) \) returns \( -\infty \).
- \( y0(x) \) returns NaN for \( x < 0 \).
- \( y0(+\infty) \) returns +0.
- \( y0(\text{NaN}) \) returns NaN.
Description
Calculate the value of the Bessel function of the second kind of order 0 for the input argument \( x \), \( Y_0(x) \).

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

__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(±0) returns \(-\infty\).
- y1(x) returns NaN for \( x < 0 \).
- y1(±\infty) returns +0.
- y1(NaN) returns NaN.

Description
Calculate the value of the Bessel function of the second kind of order 1 for the input argument \( x \), \( Y_1(x) \).

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

__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, x] returns NaN for \( n < 0 \).
- yn[n, ±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.
\texttt{\_\_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.

\textbf{Description}

Calculate the maximum value of the arguments \texttt{a} and \texttt{b}.

\texttt{\_\_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.

\textbf{Description}

Calculate the maximum value of the arguments \texttt{a} and \texttt{b}.

\texttt{\_\_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.

\textbf{Description}

Calculate the maximum value of the arguments \texttt{a} and \texttt{b}, perform integer promotion first.

\texttt{\_\_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.

\textbf{Description}

Calculate the maximum value of the arguments \texttt{a} and \texttt{b}, perform integer promotion first.

\texttt{\_\_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.

\textbf{Description}

Calculate the maximum value of the arguments \texttt{a} and \texttt{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.
\textbf{Note:}

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

\texttt{\_device\_ float \_fdiv\_ru (float x, float y)}

Divide two floating-point values in round-up mode.

\textbf{Returns}

Returns $x / y$.

\textbf{Description}

Divide two floating-point values $x$ by $y$ in round-up (to positive infinity) mode.

\textbf{Note:}

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

\texttt{\_device\_ float \_fdiv\_rz (float x, float y)}

Divide two floating-point values in round-towards-zero mode.

\textbf{Returns}

Returns $x / y$.

\textbf{Description}

Divide two floating-point values $x$ by $y$ in round-towards-zero mode.

\textbf{Note:}

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

\texttt{\_device\_ float \_fdividef (float x, float y)}

Calculate the fast approximate division of the input arguments.

\textbf{Returns}

Returns $x / y$. 
- \texttt{__fdividef(\infty, y)} returns NaN for $2^{126} < |y| < 2^{128}$.
- \texttt{__fdividef(x, y)} returns 0 for $2^{126} < |y| < 2^{128}$ and finite x.

\textbf{Description}

Calculate the fast approximate division of $x$ by $y$.

\textbf{Note:}

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Intrinsic Functions section.

\texttt{__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.

\textbf{Description}

Behavior is the same as \texttt{__fmaf\_rd(x, y, z)}, the difference is in handling denormalized inputs and outputs: -ftz compiler flag has no effect.

\texttt{__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.

\textbf{Description}

Behavior is the same as \texttt{__fmaf\_rn(x, y, z)}, the difference is in handling denormalized inputs and outputs: -ftz compiler flag has no effect.

\texttt{__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.

\textbf{Description}

Behavior is the same as \texttt{__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.
- \( fmaf(x, y, -\infty) \) returns NaN if \( xy \) is an exact \(+\infty\).
- \( fmaf(x, y, +\infty) \) 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-to-nearest-even mode.

**Note:**
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

\[
\text{__device__ float __fmaf_ru (float x, float y, float z)}
\]

Compute \( xy + z \) as a single operation, in round-up mode.

**Returns**

Returns the rounded value of \( xy + z \) as a single operation.
- \( fmaf(\pm\infty, 0, z) \) returns NaN.
- \( fmaf(0, \pm\infty, z) \) returns NaN.
- \( fmaf(x, y, -\infty) \) returns NaN if \( xy \) is an exact \(+\infty\).
- \( fmaf(x, y, +\infty) \) 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-up (to positive infinity) mode.

**Note:**
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

\[
\text{__device__ float __fmaf_rz (float x, float y, float z)}
\]

Compute \( xy + z \) as a single operation, in round-towards-zero mode.

**Returns**

Returns the rounded value of \( xy + z \) as a single operation.
- \( fmaf(\pm\infty, 0, z) \) 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\( -\infty\).

\textbf{Description}

Computes the value of\( xy + z\) as a single ternary operation, rounding the result once in round-towards-zero mode.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|}
\hline
\textbf{Note:} & \\
\hline
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section. & \\
\hline
\end{tabular}
\end{table}

\textbf{__device__ float __fmul_rd (float x, float y)}

Multiply two floating-point values in round-down mode.

\textbf{Returns}

Returns \(x \times y\).

\textbf{Description}

Compute the product of \(x\) and \(y\) in round-down (to negative infinity) mode.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|}
\hline
\textbf{Note:} & \\
\hline
- 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. & \\
\hline
\end{tabular}
\end{table}

\textbf{__device__ float __fmul_rn (float x, float y)}

Multiply two floating-point values in round-to-nearest-even mode.

\textbf{Returns}

Returns \(x \times y\).

\textbf{Description}

Compute the product of \(x\) and \(y\) in round-to-nearest-even mode.
__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.

__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.
__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 \([+0.0, 1.0]\).

`__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.

`__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.

```c
__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 \( \geq 2.0 \).

```c
__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 \(x \times y\).
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.

```
__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.

```
__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 \( x \times y \).

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.

\[ \text{__device__ double \_dsqrt\_rd (double x)} \]
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.

\[ \text{__device__ double \_dsqrt\_rn (double x)} \]
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.
__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.

**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_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.

**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 __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.

- \( \text{fma}(\pm \infty, \pm 0, z) \) returns NaN.
- \( \text{fma}(\pm 0, \pm \infty, z) \) 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-down (to negative infinity) mode.

\textbf{Note:}
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

\texttt{\_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.

\begin{itemize}
\item fmaf($\pm \infty$, $\pm 0$, $z$) returns NaN.
\item fmaf($\pm 0$, $\pm \infty$, $z$) returns NaN.
\item fmaf($x$, $y$, $-\infty$) returns NaN if $x \times y$ is an exact $+\infty$
\item fmaf($x$, $y$, $+\infty$) returns NaN if $x \times y$ is an exact $-\infty$
\end{itemize}

Description
Computes the value of $x \times y + z$ as a single ternary operation, rounding the result once in round-to-nearest-even mode.

\textbf{Note:}
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

\texttt{\_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(±∞, ±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, Double-Precision Floating-Point Functions section.

__device__ double __fma_rz(double x, double y, double 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.
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-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 \textit{x}. i.e. bit \textit{N} of the return value corresponds to bit 31-\textit{N} of \textit{x}.

Description
Reverses the bit order of the 32-bit unsigned integer \textit{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 \textit{x}. i.e. bit \textit{N} of the return value corresponds to bit 63-\textit{N} of \textit{x}.

Description
Reverses the bit order of the 64-bit unsigned integer \textit{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 \textit{x} and \textit{y}, as specified by a selector, \textit{s}.

Description
Create 8-byte source

\begin{itemize}
  \item uint64_t tmp64 = ((uint64_t)y << 32) | x;
\end{itemize}
Extract selector bits

- `selector0 = (s >> 0) & 0x7;`
- `selector1 = (s >> 4) & 0x7;`
- `selector2 = (s >> 8) & 0x7;`
- `selector3 = (s >> 12) & 0x7;`

Return 4 selected bytes from 8-byte source:

- `res[07:00] = tmp64[selector0];`
- `res[15:08] = tmp64[selector1];`
- `res[23:16] = tmp64[selector2];`
- `res[31:24] = 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__ 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.

**__device__ int __ffsll (long long int 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.

__ffsll(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.

**__device__ unsigned int __funnelshift_l (unsigned int lo, unsigned int hi, unsigned int shift)**
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 $hi$ 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 clamped value of $shift$ ($\text{min}(shift, 32)$). The most 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.

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_rc (unsigned int $lo$, unsigned int $hi$, unsigned int $shift$)
Concatenate $hi : lo$, shift right by $\text{min}(shift, 32)$ 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 $shift$ ($\text{min}(shift, 32)$). 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 \( (x + y) \gg 1 \), avoiding overflow in the intermediate sum.

\[
\texttt{__device__ int __mul24 (int x, int y)}
\]

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.

\[
\texttt{__device__ long long int __mul64hi (long long int x, long long int y)}
\]

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.

\[
\texttt{__device__ int __mulhi (int x, int y)}
\]

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 \( x \).

__device__ int __rhadd (int x, int y)
Compute rounded average of signed input arguments, avoiding overflow in the intermediate sum.

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 \( (x + y + 1) >> 1 \), avoiding overflow in the intermediate sum.

__device__ unsigned int __sad (int x, int y, unsigned int 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 $x$ and $y$ are signed 32-bit integers, input $z$ is a 32-bit unsigned integer.

__device__ unsigned int __uhadd (unsigned int x, unsigned int y)
Compute average of unsigned input arguments, avoiding overflow in the intermediate sum.

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 $(x + y) >> 1$, 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.

\[
\text{__device__ unsigned int __umulhi (unsigned int } x, \text{ unsigned int } y)\]
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.

\[
\text{__device__ unsigned int __urhadd (unsigned int } x, \text{ unsigned int } y)\]
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.

\[
\text{__device__ unsigned int __usad (unsigned int } x, \text{ unsigned int } y, \text{ unsigned int } z)\]
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 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.

__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.

__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.
__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.

```c
__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.

```c
__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.

```c
__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.

```c
__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.

__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.

__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.
__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.
\textbf{__device__ int __float2int_rd (float x)}

Convert a float to a signed integer in round-down mode.

\textbf{Returns}
Returns converted value.

\textbf{Description}
Convert the single-precision floating-point value \texttt{x} to a signed integer in round-down (to negative infinity) mode.

\textbf{__device__ int __float2int_rn (float x)}

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

\textbf{Returns}
Returns converted value.

\textbf{Description}
Convert the single-precision floating-point value \texttt{x} to a signed integer in round-to-nearest-even mode.

\textbf{__device__ int __float2int_ru (float)}

Convert a float to a signed integer in round-up mode.

\textbf{Returns}
Returns converted value.

\textbf{Description}
Convert the single-precision floating-point value \texttt{x} to a signed integer in round-up (to positive infinity) mode.

\textbf{__device__ int __float2int_rz (float x)}

Convert a float to a signed integer in round-towards-zero mode.

\textbf{Returns}
Returns converted value.
Description
Convert the single-precision floating-point value $x$ to a signed integer in round-towards-zero mode.

\[
\text{\_\_device\_\_ long long int \_\_float2ll\_rd (float x)}
\]
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.

\[
\text{\_\_device\_\_ long long int \_\_float2ll\_rn (float x)}
\]
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.

\[
\text{\_\_device\_\_ long long int \_\_float2ll\_ru (float x)}
\]
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.

\[\text{__device__ unsigned int \_\_float2uint\_rz (float x)}\]

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.

\[\text{__device__ unsigned long long int \_\_float2ull\_rd (float x)}\]

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.

\[\text{__device__ unsigned long long int \_\_float2ull\_rn (float x)}\]

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.

\[
\text{__device__ double __hiloint2double (int hi, int lo)}
\]

Reinterpret high and low 32-bit integer values as a double.

Returns
Returns reinterpreted value.

Description
Reinterpret the integer value of \( \text{hi} \) as the high 32 bits of a double-precision floating-point value and the integer value of \( \text{lo} \) as the low 32 bits of the same double-precision floating-point value.

\[
\text{__device__ double __int2double_rn (int x)}
\]

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.

\[
\text{__device__ float __int2float_rd (int x)}
\]

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.
__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.

\[
\text{\texttt{\_device\_double \_ll2double\_rd (long long int x)}}
\]
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.

\[
\text{\texttt{\_device\_double \_ll2double\_rn (long long int x)}}
\]
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.

\[
\text{\texttt{\_device\_double \_ll2double\_ru (long long int x)}}
\]
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.
__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.

Description
Convert the signed 64-bit integer value \( x \) to a double-precision floating-point value in round-towards-zero mode.

__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.

\[
\text{__device__ float __uint_as_float (unsigned int x)}
\]
Reinterpret bits in an unsigned integer as a float.

Description
Reinterpret the bits in the unsigned integer value \( x \) as a single-precision floating-point value.

\[
\text{__device__ double __ull2double_rd (unsigned long long int x)}
\]
Convert an unsigned 64-bit int to a double in round-down mode.

Description
Convert the unsigned 64-bit integer value \( x \) to a double-precision floating-point value in round-down (to negative infinity) mode.

\[
\text{__device__ double __ull2double_rn (unsigned long long int x)}
\]
Convert an unsigned 64-bit int to a double in round-to-nearest-even mode.

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.

__device__ double __ull2double_rz (unsigned long long int x)
Convert an unsigned 64-bit int to a double in round-towards-zero mode.

Returns
Returns converted value.

Description
Convert the unsigned 64-bit integer value \( x \) to a double-precision floating-point value in round-towards-zero mode.

__device__ float __ull2float_rd (unsigned long long 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 __ull2float_rn (unsigned long long 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 __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 (to positive infinity) 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 __vabsss2 (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.
\textbf{\_device\_ unsigned int \_vcmpeq2 (unsigned int a, unsigned int b)}
Performs per-halfword (un)signed comparison.

\textbf{Returns}
Returns 0xffff computed value.

\textbf{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.

\textbf{\_device\_ unsigned int \_vcmpeq4 (unsigned int a, unsigned int b)}
Performs per-byte (un)signed comparison.

\textbf{Returns}
Returns 0xff if a = b, else returns 0.

\textbf{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.

\textbf{\_device\_ unsigned int \_vcmpges2 (unsigned int a, unsigned int b)}
Performs per-halfword signed comparison: a >= b ? 0xffff : 0.

\textbf{Returns}
Returns 0xffff if a >= b, else returns 0.

\textbf{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 >= 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 __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 ? 0xffffffff : 0.

Returns
Returns 0xffffffff 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 <= 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 __vcmpleu4(0x1234aba5, 0x1234aba6) returns 0xffffffff.

__device__ unsigned int __vcmplts2 (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 __vcmplts2(0x1234aba5, 0x1234aba6) returns 0x0000ffff.
__device__ unsigned int __vcmplts4 (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 __vcmplts4(0x1234aba5, 0x1234aba6) returns 0x000000ff.

__device__ unsigned int __vcmpltu2 (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 __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 != 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 __vcmplts2(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 __vcmplts4(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: \( \max(\min(a + b, c), 0) \) Partial results are recombined and returned as unsigned int.

\[ \text{__host____device__ int __viaddmin_s32 (const int \ a, const int \ b, const int \ c)} \]

Computes \( \min(a + b, c) \).

**Returns**

Returns computed value.

**Description**

Calculates the sum of signed integers \( a \) and \( b \) and takes the min with \( c \).

\[ \text{__host____device__ int __viaddmin_s32_relu (const int \ a, const int \ b, const int \ c)} \]

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.

\[ \text{__host____device__ unsigned int __viaddmin_u16x2 (const unsigned int \ a, const unsigned int \ b, const unsigned int \ c)} \]

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: \( \min(a + b, c), c \) 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 recombed 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 \(\text{pred}\) to the value \((a \geq b)\).

___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 \(\text{pred}_\text{hi}\) and \(\text{pred}_\text{lo}\) to the per-halfword result of \((a \geq b)\).

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 \(\text{pred}_\text{hi}\) to the value \((a\_high\_part \geq b\_high\_part)\). Sets the value pointed to by \(\text{pred}_\text{lo}\) to the value \((a\_low\_part \geq b\_low\_part)\).

___host____device___ unsigned int ___vibmax_u32
(const unsigned int a, const unsigned int b, const bool *pred)
Computes \(\max(a, b)\), also sets the value pointed to by \(\text{pred}\) 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 \(\text{pred}\) to the value \((a \geq b)\).
/__host__/__device__/ unsigned int __vibmin_s16x2
(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 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 __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 <= b).

__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(max(a_part, b_part), c_part)). Partial results are recombined and returned as unsigned int.
__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 with relu ( = max(a_part, b_part, c_part, 0) ). Partial results are recombined and returned as unsigned int.

__host____device__ int __vimax3_s32 (const int a, const int b, const int c)
Computes max(max(a, b), c).

Returns
Returns computed value.

Description
Calculates the 3-way max of signed integers a, b and c.

__host____device__ int __vimax3_s32_relu (const int a, const int b, const int c)
Computes max(max(max(a, b), c), 0).

Returns
Returns computed value.

Description
Calculates the maximum of three signed ints, if this is less than 0 then 0 is returned.
__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).

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 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 \( = \max\{\min(a_{\text{part}}, b_{\text{part}}, c_{\text{part}}), 0\} \). Partial results are recombined and returned as unsigned int.

\[
\text{__host____device__ int __vimin3_s32 (const int a, const int b, const int c)}
\]

Computes \( \min\{\min(a, b), c\} \).

Returns
Returns computed value.

Description
Calculates the 3-way min of signed integers \( a \), \( b \) and \( c \).

\[
\text{__host____device__ int __vimin3_s32_relu (const int a, const int b, const int c)}
\]

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.

\[
\text{__host____device__ unsigned int __vimin3_u16x2 (const unsigned int a, const unsigned int b, const unsigned int c)}
\]

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 recombined 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)
Perform 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 __vsetltsls4 (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:

- __nv_fp8_e4m3
datatype __nv_fp8_e4m3
dataype
- __nv_fp8_e5m2
datatype __nv_fp8_e5m2
dataype
- __nv_fp8x2_e4m3
datatype __nv_fp8x2_e4m3
dataype
- __nv_fp8x2_e5m2
datatype __nv_fp8x2_e5m2
dataype
- __nv_fp8x4_e4m3
datatype __nv_fp8x4_e4m3
dataype
- __nv_fp8x4_e5m2
datatype __nv_fp8x4_e5m2
dataype

2.1. __nv_fp8_e4m3 Struct Reference

__nv_fp8_e4m3
dataype

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
cppOperationVisibility: visibility=public
cppOperationInline: inline=inline
cppConstructorExplicit: explicit=explicit
__host__ __device__
cppOperationConst: const=const
long long int val
Constructor from `long long int` data type, relies on `__NV_SATFINITE` behavior for out-of-range values.

```
__nv_fp8_e4m3
__nv_fp8_e4m3 cppOperationVisibility: visibility=public cppOperationInline: inline=inline
cppConstructorExplicit: explicit=explicit __host__ __device__ cppOperationConst:
const=const cppOperationPrimitive: storage=int  val
```

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

```
__nv_fp8_e4m3
__nv_fp8_e4m3 cppOperationVisibility: visibility=public cppOperationInline: inline=inline
cppConstructorExplicit: explicit=explicit __host__ __device__ cppOperationConst:
const=const short int  val
```

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

```
__nv_fp8_e4m3
__nv_fp8_e4m3 cppOperationVisibility: visibility=public cppOperationInline: inline=inline
cppConstructorExplicit: explicit=explicit __host__ __device__ cppOperationConst:
const=const unsigned long long int  val
```

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

```
__nv_fp8_e4m3
__nv_fp8_e4m3 cppOperationVisibility: visibility=public cppOperationInline: inline=inline
cppConstructorExplicit: explicit=explicit __host__ __device__ cppOperationConst:
const=const unsigned int  val
```

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

```
__nv_fp8_e4m3
__nv_fp8_e4m3 cppOperationVisibility: visibility=public cppOperationInline: inline=inline
cppConstructorExplicit: explicit=explicit __host__ __device__ cppOperationConst:
const=const unsigned short int  val
```

Constructor from `unsigned short int` data type, relies on `__NV_SATFINITE` behavior for out-of-range values.
__nv_fp8_e4m3
__nv_fp8_e4m3 cppOperationVisibility: visibility=public cppOperationInline: inline=inline
cppConstructorExplicit: explicit=explicit __host__ __device__  cppOperationConst:
const=const cppOperationPrimitive: storage=double  f
Constructor from double data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8_e4m3
__nv_fp8_e4m3 cppOperationVisibility: visibility=public cppOperationInline: inline=inline
cppConstructorExplicit: explicit=explicit __host__ __device__  cppOperationConst:
const=const cppOperationPrimitive: storage=float  f
Constructor from float data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8_e4m3
__nv_fp8_e4m3 cppOperationVisibility: visibility=public cppOperationInline: inline=inline
cppConstructorExplicit: explicit=explicit __host__ __device__  cppOperationConst:
const=const __nv_bfloat16  f
Constructor from __nv_bfloat16 data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8_e4m3
__nv_fp8_e4m3 cppOperationVisibility: visibility=public cppOperationInline: inline=inline
cppConstructorExplicit: explicit=explicit __host__ __device__  cppOperationConst:
const=const __half  f
Constructor from __half data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8_e4m3
__nv_fp8_e4m3 cppOperationVisibility: visibility=public
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 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 long int ()

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

__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 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.

2.2.  __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 cppOperationVisibility: visibility=public cppOperationInline: inline=inline
cppConstructorExplicit: explicit=explicit __host__ __device__ cppOperationConst:
const=const long long int  val

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

__nv_fp8_e5m2
__nv_fp8_e5m2 cppOperationVisibility: visibility=public cppOperationInline: inline=inline
cppConstructorExplicit: explicit=explicit __host__ __device__ cppOperationConst:
const=const int  val

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

__nv_fp8_e5m2
__nv_fp8_e5m2 cppOperationVisibility: visibility=public cppOperationInline: inline=inline
cppConstructorExplicit: explicit=explicit __host__ __device__ cppOperationConst:
const=const short int  val

Constructor from short int data type.

__nv_fp8_e5m2
__nv_fp8_e5m2 cppOperationVisibility: visibility=public cppOperationInline: inline=inline
cppConstructorExplicit: explicit=explicit __host__ __device__ cppOperationConst:
const=const unsigned long long int  val

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

__nv_fp8_e5m2
__nv_fp8_e5m2 cppOperationVisibility: visibility=public cppOperationInline: inline=inline
cppConstructorExplicit: explicit=explicit __host__ __device__ cppOperationConst:
const=const unsigned int  val

Constructor from unsigned int data type, relies on __NV_SATFINITE behavior for out-of-range values.
__nv_fp8_e5m2
__nv_fp8_e5m2 cppOperationVisibility: visibility=public cppOperationInline: inline=inline
cppConstructorExplicit: explicit=explicit __host__ __device__ cppOperationConst:
const=const unsigned short int  val
Constructor from unsigned short int data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8_e5m2
__nv_fp8_e5m2 cppOperationVisibility: visibility=public cppOperationInline: inline=inline
cppConstructorExplicit: explicit=explicit __host__ __device__ cppOperationConst:
const=const cppOperationPrimitive: storage=double  f
Constructor from double data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8_e5m2
__nv_fp8_e5m2 cppOperationVisibility: visibility=public cppOperationInline: inline=inline
cppConstructorExplicit: explicit=explicit __host__ __device__ cppOperationConst:
const=const cppOperationPrimitive: storage=float  f
Constructor from float data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8_e5m2
__nv_fp8_e5m2 cppOperationVisibility: visibility=public cppOperationInline: inline=inline
cppConstructorExplicit: explicit=explicit __host__ __device__ cppOperationConst:
const=const __nv_bfloat16  f
Constructor from __nv_bfloat16 data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8_e5m2
__nv_fp8_e5m2 cppOperationVisibility: visibility=public cppOperationInline: inline=inline
cppConstructorExplicit: explicit=explicit __host__ __device__ cppOperationConst:
const=const __half  f
Constructor from __half data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8_e5m2
__nv_fp8_e5m2 cppOperationVisibility: visibility=public
Constructor by default.

```cpp
__host__ __device__ operator __half ()
```

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

```cpp
__host__ __device__ operator __nv_bfloat16 ()
```

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

```cpp
__host__ __device__ operator bool ()
```

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

```cpp
__host__ __device__ operator double ()
```

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

```cpp
__host__ __device__ operator float ()
```

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

```cpp
__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 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 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.

2.3. __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 cppOperationVisibility: visibility=public cppOperationInline: inline=inline
cppConstructorExplicit: explicit=explicit __host__ __device__ cppOperationConst:
const=const double2 f
Constructor from double2 data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8x2_e4m3
__nv_fp8x2_e4m3 cppOperationVisibility: visibility=public cppOperationInline: inline=inline
cppConstructorExplicit: explicit=explicit __host__ __device__ cppOperationConst:
const=const float2 f
Constructor from float2 data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8x2_e4m3
__nv_fp8x2_e4m3 cppOperationVisibility: visibility=public cppOperationInline: inline=inline
cppConstructorExplicit: explicit=explicit __host__ __device__ cppOperationConst:
const=const __nv_bfloat162 f
Constructor from __nv_bfloat162 data type, relies on __NV_SATFINITE behavior for out-of-range values.
__nv_fp8x2_e4m3
__nv_fp8x2_e4m3 cppOperationVisibility: visibility=public cppOperationInline: inline=inline
cppConstructorExplicit: explicit=explicit __host__ __device__  cppOperationConst: const=const __half2  f

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

__nv_fp8x2_e4m3
__nv_fp8x2_e4m3 cppOperationVisibility: visibility=public

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.

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

```cpp
__nv_fp8x2_e5m2
__nv_fp8x2_e5m2 cppOperationVisibility: visibility=public cppOperationInline: inline=inline
cppConstructorExplicit: explicit=explicit __host__ __device__
cppOperationConst: const=const float2 f
```

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

```cpp
__nv_fp8x2_e5m2
__nv_fp8x2_e5m2 cppOperationVisibility: visibility=public cppOperationInline: inline=inline
cppConstructorExplicit: explicit=explicit __host__ __device__
const=const __nv_bfloat162 f
```

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

```cpp
__nv_fp8x2_e5m2
__nv_fp8x2_e5m2 cppOperationVisibility: visibility=public cppOperationInline: inline=inline
cppConstructorExplicit: explicit=explicit __host__ __device__
const=const __half2 f
```

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

```cpp
__nv_fp8x2_e5m2
__nv_fp8x2_e5m2 cppOperationVisibility: visibility=public
```

Constructor by default.

```cpp
__host____device__operator __half2 ()
```

**Description**
Conversion operator to `__half2` data type.

```cpp
__host____device__operator float2 ()
```

**Description**
Conversion operator to `float2` data type.
2.5. __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 0xFF 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 cppOperationVisibility: visibility=public cppOperationInline: inline=inline
cppConstructorExplicit: explicit=explicit __host__ __device__ cppOperationConst: const=const double4 f

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

__nv_fp8x4_e4m3

__nv_fp8x4_e4m3 cppOperationVisibility: visibility=public cppOperationInline: inline=inline
cppConstructorExplicit: explicit=explicit __host__ __device__ cppOperationConst: const=const float4 f

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

__nv_fp8x4_e4m3

__nv_fp8x4_e4m3 cppOperationVisibility: visibility=public cppOperationInline: inline=inline
cppConstructorExplicit: explicit=explicit __host__ __device__ cppOperationConst: const=const __nv_bfloat162 flo cppOperationConst: const=const __nv_bfloat162 fhi

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

__nv_fp8x4_e4m3

__nv_fp8x4_e4m3 cppOperationVisibility: visibility=public cppOperationInline: inline=inline
cppConstructorExplicit: explicit=explicit __host__ __device__ cppOperationConst: const=const __half2 flo cppOperationConst: const=const __half2 fhi

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 cppOperationVisibility: visibility=public
Constructor by default.

__host____device__operator float4 ()

Description
Conversion operator to float4 vector data type.

2.6. __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 cppOperationVisibility: visibility=public cppOperationInline: inline=inline
cppConstructorExplicit: explicit=explicit __host__ __device__ cppOperationConst:
const=const double4 f
Constructor from double4 vector data type, relies on __NV_SATFINITE behavior for out-of-range values.

__nv_fp8x4_e5m2
__nv_fp8x4_e5m2 cppOperationVisibility: visibility=public cppOperationInline: inline=inline
cppConstructorExplicit: explicit=explicit __host__ __device__ cppOperationConst:
const=const float4 f
Constructor from float4 vector data type, relies on __NV_SATFINITE behavior for out-of-range values.
__nv_fp8x4_e5m2

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

__nv_fp8x4_e5m2

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

__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:

__nv_fp8_e4m3()
__nv_fp8_e5m2()
__nv_fp8x2_e4m3()
__nv_fp8x2_e5m2()
__nv_fp8x4_e4m3()
__nv_fp8x4_e5m2()
__x
__nv_fp8x2_e5m2
__nv_fp8x4_e5m2
__nv_fp8_e5m2
__nv_fp8_e4m3
__nv_fp8x2_e4m3
__nv_fp8x4_e4m3

operator __half()  
__nv_fp8_e5m2
__nv_fp8_e4m3

operator __half2()  
__nv_fp8x2_e4m3
__nv_fp8x2_e5m2

operator __nv_bfloat16()  
__nv_fp8_e5m2


__nv_fp8_e4m3
operator bool()
    __nv_fp8_e4m3
    __nv_fp8_e5m2
operator double()  
    __nv_fp8_e5m2  
    __nv_fp8_e4m3
operator float() 
    __nv_fp8_e5m2  
    __nv_fp8_e4m3
operator float2() 
    __nv_fp8x2_e5m2  
    __nv_fp8x2_e4m3
operator float4() 
    __nv_fp8x4_e4m3  
    __nv_fp8x4_e5m2
operator int()  
    __nv_fp8_e5m2  
    __nv_fp8_e4m3
operator long long int() 
    __nv_fp8_e5m2  
    __nv_fp8_e4m3
operator short int() 
    __nv_fp8_e5m2  
    __nv_fp8_e4m3
operator signed char()  
    __nv_fp8_e5m2  
    __nv_fp8_e4m3
operator unsigned char() 
    __nv_fp8_e4m3  
    __nv_fp8_e5m2
operator unsigned int() 
    __nv_fp8_e5m2  
    __nv_fp8_e4m3
operator unsigned long long int() 
    __nv_fp8_e4m3  
    __nv_fp8_e5m2
operator unsigned short int() 
    __nv_fp8_e5m2  
    __nv_fp8_e4m3
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

© 2007-2022 NVIDIA Corporation & affiliates. All rights reserved.