CUDA Math API

API Reference Manual
# Table of Contents

## Chapter 1. Modules

1.1. 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

### 1.1.1. Half Arithmetic Functions

- `__habs`
- `__hadd`
- `__hadd_sat`
- `__hdiv`
- `__hfma`
- `__hfma_relu`
- `__hfma_sat`
- `__hmul`
- `__hmul_sat`
- `__hneg`
- `__hsub`
- `__hsub_sat`

### 1.1.2. Half2 Arithmetic Functions

- `__h2div`
- `__habs2`
- `__hadd2`
- `__hadd2_sat`
- `__hcmadd`
- `__hfma2`
- `__hfma2_relu`
- `__hfma2_sat`
- `__hmul2`
- `__hmul2_sat`
- `__hneg2`
- `__hsub2`
- `__hsub2_sat`
1.1.3. Half Comparison Functions

- __heq__
- __hequ__
- __hge__
- __hgeu__
- __hgt__
- __hgtu__
- __hisinf__
- __hisnan__
- __hle__
- __hleu__
- __hlt__
- __hltu__
- __hmax__
- __hmax_nan__
- __hmin__
- __hmin_nan__
- __hne__
- __hneu__

1.1.4. Half2 Comparison Functions

- __hbeq2__
- __hbequ2__
- __hbge2__
- __hbgeu2__
- __hbgt2__
- __hbgtu2__
- __hble2__
- __hbleu2__
- __hblt2__
- __hbltu2__
- __hbne2__
- __hbneu2__
- __heq2__
- __hequ2__
- __hge2__
- __hgeu2__
- __hgt2__
- __hgtu2__
1.1.5. Half Precision Conversion And Data Movement

- __double2half................................................................. 34
- __float22half2_rn......................................................... 35
- __float2half................................................................. 36
- __float2half2_rn........................................................... 36
- __float2half_rd............................................................. 37
- __float2half_ru............................................................. 37
- __float2half_rz............................................................. 38
- __floats2half2_rn.......................................................... 38
- __half22float2........................................................................ 39
- __half2float................................................................. 39
- __half2half................................................................. 39
- __half2int_rd............................................................... 40
- __half2int_rn............................................................... 40
- __half2int_ru............................................................... 41
- __half2int_rz............................................................... 41
- __half2ll_rd................................................................. 42
- __half2ll_rn................................................................. 42
- __half2ll_ru................................................................. 42
- __half2ll_rz................................................................. 43
- __half2short_rd............................................................ 43
- __half2short_rn............................................................ 44
- __half2short_ru............................................................ 44
- __half2short_rz............................................................ 45
- __half2uint_rd............................................................. 45
- __half2uint_rn............................................................. 45
1.1.6. Half Math Functions ................................................................. 78
  hceil ................................................................. 78
  hcos ............................................................ 78
  hexp ............................................................ 79
  hexp10 ......................................................... 79
  hexp2 ........................................................ 79
  hfloor .......................................................... 80
  hlog ............................................................ 80
  hlog10 ......................................................... 81
  hlog2 .......................................................... 81
  hrcp ............................................................ 82
  hrint ............................................................ 82
  hsqrt ........................................................... 82
  hsin ............................................................. 83
  hsqrt ........................................................... 83
  htrunc ......................................................... 84

1.1.7. Half2 Math Functions .......................................................... 84
  h2ceil .......................................................... 84
  h2cos .......................................................... 85
  h2exp .......................................................... 85
  h2exp10 ...................................................... 85
  h2exp2 ........................................................ 86
  h2floor ........................................................ 86
  h2log ........................................................... 87
  h2log10 ...................................................... 87
  h2log2 ........................................................ 88
  h2rcp ........................................................... 88
  h2rint ........................................................ 88
  h2rsqrt ....................................................... 89
  h2sin .......................................................... 89
  h2sqrt ......................................................... 90
  h2trunc ....................................................... 90

1.2. Bfloat16 Precision Intrinsics .................................................... 91
  Bfloat16 Arithmetic Functions .................................................. 91
  Bfloat162 Arithmetic Functions ............................................... 91
  Bfloat16 Comparison Functions ............................................... 91
  Bfloat162 Comparison Functions ............................................. 91
  Bfloat16 Precision Conversion And Data Movement ................... 91
Bfloat16 Math Functions............................................................................................................ 91
Bfloat162 Math Functions........................................................................................................... 91
1.2.1. Bfloat16 Arithmetic Functions......................................................................................... 91
    __h2div ............................................................................................................................. 91
    __habs ............................................................................................................................ 92
    __hadd ............................................................................................................................ 92
    __hadd_sat ....................................................................................................................... 92
    __hdiv ............................................................................................................................. 93
    __hfma ........................................................................................................................... 93
    __hfma_relu ..................................................................................................................... 93
    __hfma_sat ....................................................................................................................... 94
    __hmul ............................................................................................................................. 94
    __hmul_sat ....................................................................................................................... 95
    __hneg ............................................................................................................................ 95
    __hsub ............................................................................................................................ 95
    __hsub_sat ....................................................................................................................... 96
1.2.2. Bfloat162 Arithmetic Functions ....................................................................................... 96
    __habs2 ........................................................................................................................... 96
    __hadd2 ........................................................................................................................... 97
    __hadd2_sat ..................................................................................................................... 97
    __hcmaadd ....................................................................................................................... 98
    __hfma2 ........................................................................................................................... 98
    __hfma2_relu ................................................................................................................... 98
    __hfma2_sat ..................................................................................................................... 99
    __hmul2 ............................................................................................................................ 100
    __hmul2_sat ..................................................................................................................... 100
    __hneg2 ........................................................................................................................... 101
    __hsub2 ........................................................................................................................... 101
    __hsub2_sat ..................................................................................................................... 101
1.2.3. Bfloat16 Comparison Functions...................................................................................... 101
    __heq .............................................................................................................................. 102
    __hequ ............................................................................................................................ 102
    __hge .............................................................................................................................. 103
    __hgeu ............................................................................................................................ 103
    __hgt ................................................................................................................................ 104
    __hgtu ................................................................................................................................ 104
    __hisinf ............................................................................................................................ 105
    __hisnan .......................................................................................................................... 105
1.2.4. Bfloat16 Comparison Functions

__hbeq2................................................................. 110
__hbequ2............................................................... 110
__hbge2.................................................................... 111
__hbgeu2................................................................... 112
__hbg2................................................................. 112
__hbgtu2............................................................... 113
__hble2................................................................. 113
__hbleu2............................................................... 114
__hblt2................................................................. 115
__hbltu2............................................................... 115
__hbne2................................................................. 116
__hbneu2............................................................... 116
__heq2................................................................. 117
__hequ2............................................................... 118
__hge2.................................................................... 118
__hgeu2............................................................... 119
__hgt2................................................................. 119
__hgtu2............................................................... 120
__hisnan2............................................................. 120
__hle2................................................................. 121
__hleu2............................................................... 121
__hlt2................................................................. 122
__hltu2............................................................... 122
__hmax2.............................................................. 123
__hmax2_nan......................................................... 123
__hmin2.............................................................. 124
__hmin2_nan......................................................... 124

CUDA Math API
1.2.5. Bfloat16 Precision Conversion And Data Movement

__bfloat162float2................................................................................................................. 125
__bfloat162bfloat162............................................................................................................. 126
__bfloat162float..................................................................................................................... 126
__bfloat162int_rd...................................................................................................................127
__bfloat162int_rn...................................................................................................................127
__bfloat162int_ru...................................................................................................................128
__bfloat162int_rz...................................................................................................................128
__bfloat162ll_rd.....................................................................................................................129
__bfloat162ll_rn.....................................................................................................................129
__bfloat162ll_ru.....................................................................................................................129
__bfloat162ll_rz..................................................................................................................... 130
__bfloat162short_rd.............................................................................................................. 130
__bfloat162short_rn.............................................................................................................. 131
__bfloat162short_ru.............................................................................................................. 131
__bfloat162short_rz.............................................................................................................. 132
__bfloat162uint_rd................................................................................................................ 132
__bfloat162uint_rn................................................................................................................ 133
__bfloat162uint_ru................................................................................................................ 133
__bfloat162uint_rz.................................................................................................................134
__bfloat162ull_rd...................................................................................................................134
__bfloat162ull_rn...................................................................................................................135
__bfloat162ull_ru...................................................................................................................135
__bfloat162ull_rz...................................................................................................................136
__bfloat162ushort_rd............................................................................................................ 136
__bfloat162ushort_rn............................................................................................................137
__bfloat162ushort_ru............................................................................................................137
__bfloat162ushort_rz............................................................................................................ 138
__bfloat16_as_short..............................................................................................................138
__bfloat16_as_ushort............................................................................................................139
__double2bfloat16................................................................................................................. 139
__float22bfloat16.................................................................................................................. 140
__float2bfloat16..................................................................................................................... 140
__float2bfloat162_rn............................................................................................................. 141
__float2bfloat16_rd............................................................................................................... 141
__float2bfloat16_rn............................................................................................................... 142
__float2bfloat16_ru............................................................................................................... 142
__float2bfloat16_rz............................................................................................................... 142
__float22bfloat16 .................................................................
__float2bfloat16_ru............................................................................................................... 142
__float2bfloat16_rz................................................................................................................143
__floats2bfloat162_rn............................................................................................................143
__halves2bfloat162................................................................................................................144
__high2bfloat16.................................................................................................................. 144
__high2bfloat162................................................................................................................... 145
__high2float......................................................................................................................... 145
__highs2bfloat162................................................................................................................. 146
__int2bfloat16_rd...................................................................................................................146
__int2bfloat16_rn...................................................................................................................147
__int2bfloat16_ru...................................................................................................................147
__int2bfloat16_rz...................................................................................................................147
__ldca.....................................................................................................................................148
__ldca.....................................................................................................................................148
__ldcg.....................................................................................................................................148
__ldcg.....................................................................................................................................149
__ldcs.....................................................................................................................................149
__ldcs.....................................................................................................................................149
__ldcv..................................................................................................................................... 149
__ldcv..................................................................................................................................... 149
__ldg.......................................................................................................................................150
__ldg.......................................................................................................................................150
__ldlu..................................................................................................................................... 150
__ldlu..................................................................................................................................... 150
__ll2bfloat16_rd.....................................................................................................................151
__ll2bfloat16_rn.....................................................................................................................151
__ll2bfloat16_ru.....................................................................................................................152
__ll2bfloat16_rz..................................................................................................................... 152
__low2bfloat16.......................................................................................................................153
__low2bfloat162.....................................................................................................................153
__low2float............................................................................................................................ 154
__loighthigh2highlow.............................................................................................................154
__lows2bfloat162...................................................................................................................155
__shfl_down_sync...................................................................................................................155
__shfl_down_sync...................................................................................................................156
__shfl_sync.............................................................................................................................157
__shfl_sync.............................................................................................................................157
__shfl_up_sync.......................................................................................................................158
1.2.6. Bfloat16 Math Functions

hceil
hcos
hexp
hexp10
hexp2
hfloor
hlog
hlog10
hlog2 ...................................................................................................................................... 175
hrcp ........................................................................................................................................ 175
hrint ....................................................................................................................................... 176
hrsqt ..................................................................................................................................... 176
hsin ......................................................................................................................................... 177
hsqrt ....................................................................................................................................... 177
htrunc ..................................................................................................................................... 178
1.2.7. Bfloat16 Math Functions .............................................................................................. 178
h2ceil ...................................................................................................................................... 178
h2cos ...................................................................................................................................... 179
h2exp ...................................................................................................................................... 179
h2exp10 .................................................................................................................................. 179
h2exp2 .................................................................................................................................... 180
h2floor .................................................................................................................................... 180
h2log ...................................................................................................................................... 181
h2log10 ................................................................................................................................... 181
h2log2 ..................................................................................................................................... 182
h2rcp ...................................................................................................................................... 182
h2rint ...................................................................................................................................... 182
h2rsqrt ................................................................................................................................... 183
h2sin ....................................................................................................................................... 183
h2sqrt ..................................................................................................................................... 184
h2trunc ................................................................................................................................... 184
1.3. Mathematical Functions ....................................................................................................... 185
1.4. Single Precision Mathematical Functions ........................................................................... 185
acosf ........................................................................................................................................... 185
acoshf ........................................................................................................................................ 186
asin ........................................................................................................................................... 186
asinhf ......................................................................................................................................... 187
atan2f .........................................................................................................................................187
atanf ........................................................................................................................................... 187
atanhf ........................................................................................................................................ 188
cbrtf ............................................................................................................................................ 188
ceilf ............................................................................................................................................ 189
copysignf ..................................................................................................................................189
cosf ............................................................................................................................................ 189
coshf .......................................................................................................................................... 190
cosf ............................................................................................................................................ 190
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>cyl_bessel_i0f</td>
<td>191</td>
</tr>
<tr>
<td>cyl_bessel_i1f</td>
<td>191</td>
</tr>
<tr>
<td>erfcf</td>
<td>191</td>
</tr>
<tr>
<td>erfcinvf</td>
<td>192</td>
</tr>
<tr>
<td>erfxf</td>
<td>192</td>
</tr>
<tr>
<td>erff</td>
<td>193</td>
</tr>
<tr>
<td>erfinvf</td>
<td>193</td>
</tr>
<tr>
<td>exp10f</td>
<td>194</td>
</tr>
<tr>
<td>exp2f</td>
<td>194</td>
</tr>
<tr>
<td>expf</td>
<td>194</td>
</tr>
<tr>
<td>expm1f</td>
<td>195</td>
</tr>
<tr>
<td>fabsf</td>
<td>195</td>
</tr>
<tr>
<td>fdimf</td>
<td>196</td>
</tr>
<tr>
<td>fdivdef</td>
<td>196</td>
</tr>
<tr>
<td>floorf</td>
<td>197</td>
</tr>
<tr>
<td>fmaf</td>
<td>197</td>
</tr>
<tr>
<td>fmaxf</td>
<td>198</td>
</tr>
<tr>
<td>fminf</td>
<td>198</td>
</tr>
<tr>
<td>fmodf</td>
<td>198</td>
</tr>
<tr>
<td>frexpf</td>
<td>199</td>
</tr>
<tr>
<td>hypotf</td>
<td>200</td>
</tr>
<tr>
<td>ilogbf</td>
<td>200</td>
</tr>
<tr>
<td>isfinite</td>
<td>201</td>
</tr>
<tr>
<td>isinf</td>
<td>201</td>
</tr>
<tr>
<td>isnan</td>
<td>201</td>
</tr>
<tr>
<td>j0f</td>
<td>202</td>
</tr>
<tr>
<td>j1f</td>
<td>202</td>
</tr>
<tr>
<td>jnf</td>
<td>203</td>
</tr>
<tr>
<td>ldexpf</td>
<td>203</td>
</tr>
<tr>
<td>lgammaf</td>
<td>204</td>
</tr>
<tr>
<td>llrintf</td>
<td>204</td>
</tr>
<tr>
<td>llroundf</td>
<td>204</td>
</tr>
<tr>
<td>log10f</td>
<td>205</td>
</tr>
<tr>
<td>log1pf</td>
<td>205</td>
</tr>
<tr>
<td>log2f</td>
<td>206</td>
</tr>
<tr>
<td>logbf</td>
<td>206</td>
</tr>
<tr>
<td>logf</td>
<td>207</td>
</tr>
<tr>
<td>llrintf</td>
<td>207</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>lroundf</td>
<td>207</td>
</tr>
<tr>
<td>max</td>
<td>208</td>
</tr>
<tr>
<td>min</td>
<td>208</td>
</tr>
<tr>
<td>modff</td>
<td>208</td>
</tr>
<tr>
<td>nanf</td>
<td>209</td>
</tr>
<tr>
<td>nearbyintf</td>
<td>209</td>
</tr>
<tr>
<td>nextafterf</td>
<td>209</td>
</tr>
<tr>
<td>norm3df</td>
<td>210</td>
</tr>
<tr>
<td>norm4df</td>
<td>210</td>
</tr>
<tr>
<td>normcdf</td>
<td>211</td>
</tr>
<tr>
<td>normcdfinvf</td>
<td>211</td>
</tr>
<tr>
<td>normf</td>
<td>212</td>
</tr>
<tr>
<td>powf</td>
<td>212</td>
</tr>
<tr>
<td>rcbrtf</td>
<td>213</td>
</tr>
<tr>
<td>remainderf</td>
<td>213</td>
</tr>
<tr>
<td>remquof</td>
<td>214</td>
</tr>
<tr>
<td>rhypotf</td>
<td>214</td>
</tr>
<tr>
<td>rintf</td>
<td>215</td>
</tr>
<tr>
<td>rnorm3df</td>
<td>215</td>
</tr>
<tr>
<td>rnorm4df</td>
<td>216</td>
</tr>
<tr>
<td>rnormf</td>
<td>216</td>
</tr>
<tr>
<td>roundf</td>
<td>217</td>
</tr>
<tr>
<td>rsqrtf</td>
<td>217</td>
</tr>
<tr>
<td>scalblnf</td>
<td>217</td>
</tr>
<tr>
<td>scalbnf</td>
<td>218</td>
</tr>
<tr>
<td>signbit</td>
<td>218</td>
</tr>
<tr>
<td>sincosf</td>
<td>219</td>
</tr>
<tr>
<td>sincospif</td>
<td>219</td>
</tr>
<tr>
<td>sinf</td>
<td>220</td>
</tr>
<tr>
<td>sinhf</td>
<td>220</td>
</tr>
<tr>
<td>sinpif</td>
<td>221</td>
</tr>
<tr>
<td>sqrtf</td>
<td>221</td>
</tr>
<tr>
<td>tanf</td>
<td>222</td>
</tr>
<tr>
<td>tanhf</td>
<td>222</td>
</tr>
<tr>
<td>tgammaf</td>
<td>222</td>
</tr>
<tr>
<td>truncf</td>
<td>223</td>
</tr>
<tr>
<td>y0f</td>
<td>223</td>
</tr>
<tr>
<td>y1f</td>
<td>224</td>
</tr>
</tbody>
</table>
1.5. Double Precision Mathematical Functions

- acos
- acosh
- asin
- asinh
- atan
- atan2
- atanh
- cbrt
- ceil
- copysign
- cos
- cosh
- cospi
- cyl_bessel_i0
- cyl_bessel_i1
- erf
- erfc
- erfcinv
- erfcx
- erfinv
- exp
- exp10
- exp2
- expm1
- fabs
- fdim
- floor
- fma
- fmax
- fmin
- fmod
- frexp
- hypot
- ilogb
- isfinite
- isinf
isnan
j0
j1
jn
ldexp
lgamma
llrint
llround
log
log10
log1p
log2
logb
lrint
lround
max
max
max
min
min
min
modf
nan
nearbyint
nextafter
norm
norm3d
norm4d
normcdf
normcdfinv
pow
rcbrt
remainder
remquo
rhypot
rint
rnorm
rnorm3d

240
241
241
242
242
243
243
243
244
244
245
245
245
246
246
246
246
247
247
247
247
248
248
248
248
249
249
250
250
251
251
252
252
253
253
254
255
255
255
256
1.6. Integer Mathematical Functions.......................................................................................... 265
abs............................................................................................................................................. 265
labs.......................................................................................................................................... 265
llabs.......................................................................................................................................... 265
llmax.......................................................................................................................................... 265
llmin.......................................................................................................................................... 266
max........................................................................................................................................... 266
max........................................................................................................................................... 266
max........................................................................................................................................... 266
max........................................................................................................................................... 266
max........................................................................................................................................... 266
max........................................................................................................................................... 266
max........................................................................................................................................... 266
max........................................................................................................................................... 267
max........................................................................................................................................... 267
max........................................................................................................................................... 267
max........................................................................................................................................... 267
max........................................................................................................................................... 267
max........................................................................................................................................... 268
max........................................................................................................................................... 268
max........................................................................................................................................... 268
min............................................................................................................................................. 268
CUDA Math API vRelease Version | xviii
1.7. Single Precision Intrinsics

__cosf___________________________272
__exp10f________________________272
__expf___________________________272
__fadd_rd_______________________273
__fadd_rn_______________________273
__fadd_ru_______________________274
__fadd_rz_______________________274
__fdiv_rd_______________________274
__fdiv_rn_______________________275
__fdiv_ru_______________________275
__fdiv_rz_______________________276
__fdividef_______________________276
__fmaf_ieee_rd__________________277
__fmaf_ieee_rn___________________277
__fmaf_ieee_ru___________________277
__fmaf_ieee_rz___________________277
__fmaf_rd_______________________278
__fmaf_rn_______________________278
__fmaf_ru_______________________279
__fmaf_rz_______________________279
__fmul_rd_______________________280
__fmul_rn_______________________280
1.8. Double Precision Intrinsics... 

__dadd_rd.................................................................................................................................. 290
__dadd_rn.................................................................................................................................. 290
__dadd_ru.................................................................................................................................. 291
__dadd_rz.................................................................................................................................. 291
__ddiv_rd.................................................................................................................................. 292
__ddiv_rn.................................................................................................................................. 292
__ddiv_ru.................................................................................................................................. 292
__ddiv_rz.................................................................................................................................. 292
__dmul_rd.................................................................................................................................. 293
__dmul_rn.................................................................................................................................. 293
__dmul_ru.................................................................................................................................. 294
__dmul_rz.................................................................................................................................. 294
__drcp_rd.................................................................................................................................. 295
__drcp_rn.................................................................................................................................. 295
1.9. Integer Intrinsics

_brev.......................................................................................................................... 302
_brevll.......................................................................................................................... 303
_byte_perm.................................................................................................................... 303
_clz............................................................................................................................... 303
_clzll............................................................................................................................. 304
_ffs................................................................................................................................. 304
_ffsll............................................................................................................................. 304
_funnelshift_l.................................................................................................................. 305
_funnelshift_lc................................................................................................................. 305
_funnelshift_r................................................................................................................ 305
_funnelshift_rc............................................................................................................... 306
_hadd............................................................................................................................. 306
_mul24........................................................................................................................... 306
_mul64hi......................................................................................................................... 307
_mulhi............................................................................................................................. 307
_popc............................................................................................................................... 307
_popcll........................................................................................................................... 307
_rhadd............................................................................................................................ 308
_sad................................................................................................................................. 308
_uhadd............................................................................................................................. 308
_umul24.......................................................................................................................... 309
_umul64hi......................................................................................................................... 309
_umulhi............................................................................................................................ 309
1.10. Type Casting Intrinsics ....................................................................................................... 310

__double2float_rd..................................................................................................................... 311
__double2float_rn..................................................................................................................... 311
__double2float_ru..................................................................................................................... 311
__double2float_rz......................................................................................................................311
__double2hiint...........................................................................................................................312
__double2int_rd........................................................................................................................ 312
__double2int_rn........................................................................................................................ 312
__double2int_ru........................................................................................................................ 313
__double2int_rz.........................................................................................................................313
__double2ll_rd...........................................................................................................................313
__double2ll_rn.......................................................................................................................... 313
__double2ll_ru.......................................................................................................................... 314
__double2ll_rz...........................................................................................................................314
__double2loint...........................................................................................................................314
__double2uint_rd...................................................................................................................... 315
__double2uint_rn...................................................................................................................... 315
__double2uint_ru...................................................................................................................... 315
__double2uint_rz.......................................................................................................................315
__double2ull_rd........................................................................................................................ 316
__double2ull_rn........................................................................................................................ 316
__double2ull_ru........................................................................................................................ 316
__double2ull_rz...........................................................................................................................317
__double_as_longlong..............................................................................................................317
__float2int_rd............................................................................................................................ 317
__float2int_rn............................................................................................................................ 317
__float2int_ru............................................................................................................................ 318
__float2int_rz............................................................................................................................ 318
__float2ll_rd.............................................................................................................................. 318
__float2ll_rn.............................................................................................................................. 318
__float2ll_ru.............................................................................................................................. 319
__float2ll_rz...............................................................................................................................319
__float2uint_rd.......................................................................................................................... 320
__float2uint_rn.......................................................................................................................... 320
__float2uint_ru.......................................................................................................................... 320
__float2uint_rz.......................................................................................................................... 320
1.11. SIMD Intrinsics

__vabs2

__float2ull_rd
__float2ull_rn
__float2ull_ru
__float2ull_rz
__float_as_int
__float_as_uint
__hiloint2double
__int2double_rn
__int2float_rd
__int2float_rn
__int2float_ru
__int2float_rz
__int_as_float
__ll2double_rd
__ll2double_rn
__ll2double_ru
__ll2double_rz
__ll2float_rd
__ll2float_rn
__ll2float_ru
__ll2float_rz
__longlong_as_double
__uint2double_rd
__uint2double_rn
__uint2float_rd
__uint2float_rn
__uint2float_ru
__uint2float_rz
__uint_as_float
__ull2double_rd
__ull2double_rn
__ull2double_ru
__ull2double_rz
__ull2float_rd
__ull2float_rn
__ull2float_ru
__ull2float_rz

CUDA Math API

vRelease Version | xxiii
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>__vabs4</td>
<td>332</td>
</tr>
<tr>
<td>__vabsdiffs2</td>
<td>332</td>
</tr>
<tr>
<td>__vabsdiffs4</td>
<td>332</td>
</tr>
<tr>
<td>__vabsdiffu2</td>
<td>333</td>
</tr>
<tr>
<td>__vabsdiffu4</td>
<td>333</td>
</tr>
<tr>
<td>__vabsss2</td>
<td>333</td>
</tr>
<tr>
<td>__vabsss4</td>
<td>334</td>
</tr>
<tr>
<td>__vadd2</td>
<td>334</td>
</tr>
<tr>
<td>__vadd4</td>
<td>334</td>
</tr>
<tr>
<td>__vaddss2</td>
<td>335</td>
</tr>
<tr>
<td>__vaddss4</td>
<td>335</td>
</tr>
<tr>
<td>__vaddus2</td>
<td>335</td>
</tr>
<tr>
<td>__vaddus4</td>
<td>335</td>
</tr>
<tr>
<td>__vavgs2</td>
<td>336</td>
</tr>
<tr>
<td>__vavgs4</td>
<td>336</td>
</tr>
<tr>
<td>__vavgu2</td>
<td>337</td>
</tr>
<tr>
<td>__vavgu4</td>
<td>337</td>
</tr>
<tr>
<td>__vcmpeq2</td>
<td>337</td>
</tr>
<tr>
<td>__vcmpeq4</td>
<td>337</td>
</tr>
<tr>
<td>__vcmpges2</td>
<td>338</td>
</tr>
<tr>
<td>__vcmpges4</td>
<td>338</td>
</tr>
<tr>
<td>__vcmpgeu2</td>
<td>339</td>
</tr>
<tr>
<td>__vcmpgeu4</td>
<td>339</td>
</tr>
<tr>
<td>__vcmpgts2</td>
<td>339</td>
</tr>
<tr>
<td>__vcmpgts4</td>
<td>340</td>
</tr>
<tr>
<td>__vcmpgtu2</td>
<td>340</td>
</tr>
<tr>
<td>__vcmpgtu4</td>
<td>340</td>
</tr>
<tr>
<td>__vcmples2</td>
<td>341</td>
</tr>
<tr>
<td>__vcmples4</td>
<td>341</td>
</tr>
<tr>
<td>__vcmpleu2</td>
<td>341</td>
</tr>
<tr>
<td>__vcmpleu4</td>
<td>341</td>
</tr>
<tr>
<td>__vcmplts2</td>
<td>342</td>
</tr>
<tr>
<td>__vcmplts4</td>
<td>342</td>
</tr>
<tr>
<td>__vcmpltu2</td>
<td>343</td>
</tr>
<tr>
<td>__vcmpltu4</td>
<td>343</td>
</tr>
<tr>
<td>__vcmpne2</td>
<td>343</td>
</tr>
<tr>
<td>__vcmpne4</td>
<td>344</td>
</tr>
<tr>
<td>__vhaddu2</td>
<td>344</td>
</tr>
</tbody>
</table>
Chapter 1. Modules

Here is a list of all modules:

- 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
  - Bfloat16 Math Functions
  - Bfloat162 Math Functions
- Mathematical Functions
  - Single Precision Mathematical Functions
  - Double Precision Mathematical Functions
- Integer Mathematical Functions
  - Single Precision Intrinsics
  - Double Precision Intrinsics
  - Integer Intrinsics
- Type Casting Intrinsics
- SIMD Intrinsics
1.1. Half Precision Intrinsic Functions

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

1.1.1. Half Arithmetic Functions

Half Precision Intrinsic Functions

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.

__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.

__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_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_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.

1.1.2. 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_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_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_sat (const __half2 a, const __half2 b)
Performs half2 vector subtraction in round-to-nearest-even mode, with saturation to [0.0, 1.0].

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

Returns
half2
  ▶ The subtraction of vector b from a, with respect to saturation.

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

1.1.3. Half Comparison Functions
Half Precision Intrinsics
To use these functions include the header file 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.
b
- half. Is only being read.

Returns
bool

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

Description
Performs half greater-equal comparison of inputs a and b. NaN inputs generate false results.
**__device__ bool __hgeu (const __half a, const __half b)**

Performs \texttt{half} unordered greater-equal comparison.

**Parameters**

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

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

**Returns**

\texttt{bool}

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

**Description**

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

**__device__ bool __hgt (const __half a, const __half b)**

Performs \texttt{half} greater-than comparison.

**Parameters**

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

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

**Returns**

\texttt{bool}

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

**Description**

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

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

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

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

__device__ int __hisinf (const __half a)
Checks if the input float number is infinite.

Parameters
a
- float. Is only being read.

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

Description
Checks if the input float 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.
- 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.

\[
\_\text{device}\_\_\text{half}\_\_\text{hmax}\ (\text{const}\ _\text{half}\ a,\ \text{const}\ _\text{half}\ b)
\]

Calculates half maximum of two input values.

**Description**

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

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

\[
\_\text{device}\_\_\text{half}\_\_\text{hmax}_\text{nan}\ (\text{const}\ _\text{half}\ a,\ \text{const}\ _\text{half}\ b)
\]

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

**Description**

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

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

\[
\_\text{device}\_\_\text{half}\_\_\text{hmin}\ (\text{const}\ _\text{half}\ a,\ \text{const}\ _\text{half}\ b)
\]

Calculates half minimum of two input values.

**Description**

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

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

__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.

1.1.4. 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

>> trueif
    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
- trueif both half results of unordered if-equal comparison of vectors a and b are true;
- falseotherwise.

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
- trueif both half results of greater-equal comparison of vectors a and b are true;
- falseotherwise.

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 __half results of unordered 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 true results.

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

Parameters
a
- __half2. Is only being read.
b
- __half2. Is only being read.

Returns
bool
▶ true if both __half results of greater-than comparison of vectors a and b are true;
▶ false otherwise.
Description
Performs \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. \texttt{NaN} inputs generate false results.

\begin{verbatim}
__device__ bool __hbgtu2 (const __half2 a, const __half2 b)
\end{verbatim}
Performs \texttt{half2} vector unordered greater-than comparison, and returns boolean true iff both \texttt{half} results are true, boolean false otherwise.

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

Returns
\begin{verbatim}
bool
\end{verbatim}

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. \texttt{NaN} inputs generate true results.

\begin{verbatim}
__device__ bool __hble2 (const __half2 a, const __half2 b)
\end{verbatim}
Performs \texttt{half2} vector less-equal comparison, and returns boolean true iff both \texttt{half} results are true, boolean false otherwise.

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

Returns
\begin{verbatim}
bool
\end{verbatim}
true

both half results of less-equal comparison of vectors \( a \) and \( b \) are true;

false

otherwise.

Description

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

\[
_{\text{device}} \ _{\text{bool}} \ _{\text{hbleu2}} \ (\text{const } \text{half2} \ a, \text{const } \text{half2} \ b)
\]

Performs half2 vector unordered less-equal comparison, and returns boolean true iff both half results are true, boolean false otherwise.

Parameters

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

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

Returns

bool
true

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.

\[
_{\text{device}} \ _{\text{bool}} \ _{\text{hblt2}} \ (\text{const } \text{half2} \ a, \text{const } \text{half2} \ b)
\]

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 \texttt{a} and \texttt{b} are true;
- false otherwise.

Description

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

\texttt{\_\_device\_\_ bool \_\_hbltu2 (const \_\_half2 a, const \_\_half2 b)}

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

Parameters

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

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

Returns

bool

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

Description

Performs \texttt{half2} vector less-than comparison of inputs \texttt{a} and \texttt{b}. The bool result is set to true only if both \texttt{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
  both half results of not-equal comparison of vectors a and b are true,
▶ false
  otherwise.

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

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

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

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

```c
__device__ __half2 __heq2 (const __half2 a, const __half2 b)
```
Performs `half2` vector if-equal comparison.

**Parameters**

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

**Returns**

`half2` - The vector result of if-equal comparison of vectors `a` and `b`.

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

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

```c
__device__ __half2 __hge2 (const __half2 a, const __half2 b)
```
Performs half2 vector greater-equal comparison.

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

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

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

```c
__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__ __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__ __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__ __half2 __hisnan2 (const __half2 a)
Determine whether half2 argument is a NaN.

Parameters
a
- half2. Is only being read.

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

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

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

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

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

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

__device__ __half2 __hlt2 (const __half2 a, const __half2 b)
Performs half2 vector less-than comparison.

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

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

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

__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

```c
__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

```c
__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

```c
__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__ __half2 __hneu2 (const __half2 a, const __half2 b)
Performs \texttt{half2} vector unordered not-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 unordered not-equal comparison of vectors \texttt{a} and \texttt{b}.
\end{itemize}

**Description**

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

### 1.1.5. Half Precision Conversion And Data Movement

**Half Precision Intrinsics**

To use these functions include the header file \texttt{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 \texttt{half} with converted value.

**Parameters**

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

**Returns**

\texttt{half}

\begin{itemize}
  \item a converted to half.
\end{itemize}
Description
Converts double number \( a \) to half precision in round-to-nearest-even mode.

\[
\text{__host\_\_device\_\_half2 \_\_float2half2\_rn \ (const \ float2 \ a)}
\]
Converts both components of float2 number to half precision in round-to-nearest-even mode and returns \texttt{half2} with converted values.

Parameters
\( a \)
- float2. Is only being read.

Returns
\texttt{half2}
- The \texttt{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 \texttt{half2} number. Low 16 bits of the return value correspond to \( a.x \) and high 16 bits of the return value correspond to \( a.y \).

\[
\text{__host\_\_device\_\_half \_\_float2half \ (const \ float \ \ a)}
\]
Converts float number to half precision in round-to-nearest-even mode and returns \texttt{half} with converted value.

Parameters
\( a \)
- float. Is only being read.

Returns
\texttt{half}
- \( a \) converted to half.

Description
Converts float number \( a \) to half precision in round-to-nearest-even mode.
__host__ __device__ __half __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.
\texttt{\_\_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.

\textbf{Parameters}

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

\textbf{Returns}

\texttt{half}
- a converted to half.

\textbf{Description}

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

\texttt{\_\_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.

\textbf{Parameters}

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

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

\textbf{Returns}

\texttt{half2}
- The half2 value with corresponding halves equal to the converted input floats.

\textbf{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 \texttt{a}, high 16 bits correspond to the input \texttt{b}. 
__host____device__ float2 __half2float2 (const __half2 a)
Converting both halves of half2 to float2 and returns the result.

Parameters

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

Returns

float2

  a converted to float2.

Description

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

__host____device__ float __half2float (const __half a)
Converting half number to float.

Parameters

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

Returns

float

  a converted to float.

Description

Converting 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 (__half h)
Convert a half to a signed integer in round-down mode.

Parameters
h
- half. Is only being read.

Returns
int

\[ h \text{ converted to a signed integer.} \]

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

__device__ int __half2int_rn (__half h)
Convert a half to a signed integer in round-to-nearest-even mode.

Parameters
h
- half. Is only being read.

Returns
int

\[ h \text{ converted to a signed integer.} \]
Description
Convert the half-precision floating point value $h$ to a signed integer in round-to-nearest-even mode.

__device__ int __half2int_ru (__half h)
Convert a half to a signed integer in round-up mode.

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

Returns
int
├ \n├ $h$ converted to a signed integer.

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

__host__ __device__ int __half2int_rz (__half h)
Convert a half to a signed integer in round-towards-zero mode.

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

Returns
int
├ \n├ $h$ converted to a signed integer.

Description
Convert the half-precision floating point value $h$ to a signed integer in round-towards-zero mode.
__device__ long long int __half2ll_rd (__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.

__device__ long long int __half2ll_rn (__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.

__device__ long long int __half2ll_ru (__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

\[ \text{h converted to a signed 64-bit integer.} \]

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

\[
\text{__host__ __device__ long long int __half2ll_rz (__half h)}
\]
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

\[ \text{h converted to a signed 64-bit integer.} \]

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

\[
\text{__device__ short int __half2short_rd (__half h)}
\]
Convert a half to a signed short integer in round-down mode.

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

Returns
short int

\[ \text{h converted to a signed short integer.} \]
Description
Convert the half-precision floating point value \( h \) to a signed short integer in round-down mode.

\[
\text{\_device\_ short int \_half2short\_rn (__half h)}
\]
Convert a half to a signed short integer in round-to-nearest-even mode.

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

Returns
short int

\[
\text{h converted to a signed short integer.}
\]

Description
Convert the half-precision floating point value \( h \) to a signed short integer in round-to-nearest-even mode.

\[
\text{\_device\_ short int \_half2short\_ru (__half h)}
\]
Convert a half to a signed short integer in round-up mode.

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

Returns
short int

\[
\text{h converted to a signed short integer.}
\]

Description
Convert the half-precision floating point value \( h \) to a signed short integer in round-up mode.
__host__ __device__ short int __half2short_rz(__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.

__device__ unsigned int __half2uint_rd(__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.

__device__ unsigned int __half2uint_rn(__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.

\[
\textbf{__device__ unsigned int __half2uint_ru (__half h)}
\]
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.

\[
\textbf{__host__device__ unsigned int __half2uint_rz (__half h)}
\]
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.

\[
\text{__device__ unsigned long long int __half2ull_rd (__half h)}
\]
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.

\[
\text{__device__ unsigned long long int __half2ull_rn (__half h)}
\]
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.
__device__ unsigned long long int __half2ull_ru (__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

\begin{itemize}
\item h converted to an unsigned 64-bit integer.
\end{itemize}

Description
Convert the half-precision floating point value \( h \) to an unsigned 64-bit integer in round-up mode.

__host__ __device__ unsigned long long int __half2ull_rz (__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

\begin{itemize}
\item h converted to an unsigned 64-bit integer.
\end{itemize}

Description
Convert the half-precision floating point value \( h \) to an unsigned 64-bit integer in round-towards-zero mode.
__device__ unsigned short int __half2ushort_rd (__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.

__device__ unsigned short int __half2ushort_rn (__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.

__device__ unsigned short int __half2ushort_ru (__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.

\[
\text{__host____device__ unsigned short int __half2ushort_rz (__half h)}
\]
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.

\[
\text{__device__ short int __half_as_short (const __half h)}
\]
Reinterprets bits in a half as a signed short integer.

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

Returns
short int

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

\[ \textbf{__device__ unsigned short int __half_as_ushort (const __half h)} \]
Reinterprets bits in a \texttt{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.

\[ \textbf{__device__ __half2 __halves2half2 (const __half a, const __half b)} \]
Combines two \texttt{half} numbers into one \texttt{half2} number.

Parameters
\( a \)
- half. Is only being read.

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

Returns
\texttt{half2}
- The \texttt{half2} with one half equal to \( a \) and the other to \( b \).
Description
Combines two input half number \( a \) and \( b \) into one half\(_2\) 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.

\[
\_\text{host}\_\_\text{device}\_\_\text{float} \_\text{high2float} \left( \text{const} \_\_\text{half2} \; a \right)
\]
Converts high 16 bits of half\(_2\) to float and returns the result.

Parameters
\( a \)
- half\(_2\). Is only being read.

Returns
float
- The high 16 bits of \( a \) converted to float.

Description
Converts high 16 bits of half\(_2\) input \( a \) to 32 bit floating point number and returns the result.

\[
\_\text{device}\_\_\text{half} \_\text{high2half} \left( \text{const} \_\_\text{half2} \; a \right)
\]
Returns high 16 bits of half\(_2\) input.

Parameters
\( a \)
- half\(_2\). Is only being read.

Returns
half
- The high 16 bits of the input.
__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.

__device__ __half2 __highs2half2 (const __half2 a, const __half2 b)
Extracts high 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 high 16 bits of a and of b.

Description
Extracts high 16 bits from each of the two half2 inputs and combines into one half2 number. High 16 bits from input a is stored in low 16 bits of the return value, high 16 bits from input b is stored in high 16 bits of the return value.
__device__ __half __int2half_rd (int i)
Convert a signed integer to a half in round-down 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-down mode.

__host____device__ __half __int2half_rn (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 (int i)
Convert a signed integer to a half in round-up mode.

Parameters
i
- int. Is only being read.
Returns
half
  \p
  i converted to half.

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

\[ \texttt{__device__ \_half \_int2half\_rz (int i)} \]
Convert a signed integer to a half in round-towards-zero mode.

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

Returns
half
  \p
  i converted to half.

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

\[ \texttt{__device__ \_half \_ldca (const \_half *ptr)} \]
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`

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

Parameters

ptr
- memory location

Returns
The value pointed by `ptr`

__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 (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.

```c
__host__ __device__ __half __ll2half_rn (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.

```c
__device__ __half __ll2half_ru (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.
__device__ __half__ ll2half_rz (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

\[ \text{i converted to half.} \]

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 \(\text{half2}\) to float and returns the result.

Parameters
a
- half2. Is only being read.

Returns
float

\[ \text{The low 16 bits of } a \text{ converted to float.} \]

Description
Converts low 16 bits of \(\text{half2}\) input \(a\) to 32 bit floating point number and returns the result.

__device__ __half__ low2half (const __half2 a)
Returns low 16 bits of \(\text{half2}\) 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 __lowhigh2highlow (const __half2 a)
Swaps both halves of the half2 input.

Parameters
  a
  - half2. Is only being read.

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

\texttt{\_\_device\_\_half2 \_\_lows2half2 (const \_\_half2 \texttt{a}, const \_\_half2 \texttt{b})}
Extracts low 16 bits from each of the two \texttt{half2} inputs and combines into one \texttt{half2} number.

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

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

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

\texttt{\_\_device\_\_half \_\_shfl\_down\_sync (unsigned mask, \_\_half var, unsigned int delta, int width)}
Exchange a variable between threads within a warp. Copy from a thread with higher ID relative to the caller.

Parameters
\texttt{mask}
- unsigned int. Is only being read.
\texttt{var}
- half. Is only being read.
\texttt{delta}
- int. Is only being read.
\texttt{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.

__device__ __half2 __shfl_down_sync (unsigned mask, __half2 var, unsigned int delta, 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.
__device__ __half __shfl_sync (unsigned mask, __half var, int delta, 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. ithin 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__ __half2 __shfl_sync (unsigned mask, __half2 var, int delta, 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 (unsigned mask, __half var, unsigned int delta, 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...
__device__ __half2 __shfl_up_sync (unsigned mask, __half2 var, unsigned int delta, 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.

__device__ __half __shfl_xor_sync (unsigned mask, __half var, int delta, 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.

__device__ __half2 __shfl_xor_sync (unsigned mask, __half2 var, int delta, int width)
Exchange a variable between threads within a warp. Copy from a thread based on bitwise XOR of own thread ID.

Parameters

mask
- unsigned int. Is only being read.

var
- half2. Is only being read.

delta
- int. Is only being read.

width
- int. Is only being read.

Returns
Returns the 4-byte word referenced by var from the source thread ID as half2. If the source thread ID is out of range or the source thread has exited, the calling thread’s own var is returned.
Description
Calculates a source thread ID by performing a bitwise XOR of the caller’s thread ID with mask: the value of var held by the resulting thread ID is returned. If width is less than warpSize then each group of width consecutive threads are able to access elements from earlier groups of threads, however if they attempt to access elements from later groups of threads their own value of var will be returned. This mode implements a butterfly addressing pattern such as is used in tree reduction and broadcast.

__device__ __half __short2half_rd (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

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

__host____device__ __half __short2half_rn (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

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

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

\[
\text{__device__ __half __short2half_ru (short int } i \text{)}
\]

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

Parameters

\( i \)

- short int. Is only being read.

Returns

half

\[ i \text{ converted to half.} \]

Description

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

\[
\text{__device__ __half __short2half_rz (short int } i \text{)}
\]

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

Parameters

\( i \)

- short int. Is only being read.

Returns

half

\[ i \text{ converted to half.} \]

Description

Convert the signed short integer value \( i \) to a half-precision floating point value in round-towards-zero mode.
__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 (__half *ptr, __half value)
Generates a `st.global.cg` store instruction.

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

__device__ void __stcg (__half2 *ptr, __half2 value)
Generates a `st.global.cg` store instruction.

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

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

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

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

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

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

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

__device__ void __stwb (__half2 *ptr, __half2 value)
Generates a `st.global.wb` store instruction.

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

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

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

Parameters

ptr
- memory location

value
- the value to be stored

__device__ __half __uint2half_rd (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 (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.

\[
__device__ __half __uint2half_ru (unsigned int i)
\]
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.

\[
__device__ __half __uint2half_rz (unsigned int i)
\]
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.
**__device__ __half __ull2half_rd (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 (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 (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 (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 (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

\[ i \text{ converted to half.} \]

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

\[ \text{\_host\_\_device\_\_half \_\_ushort2half\_rn (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

\[ i \text{ converted to half.} \]

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

\[ \text{\_device\_\_half \_\_ushort2half\_ru (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.

\[
\text{__device__ __half __ushort2half_rz}(\text{unsigned short int } i)
\]

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

**Parameters**

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

**Returns**

half

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

**Description**

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

\[
\text{__device__ __half __ushort_as_half}(\text{const unsigned short int } i)
\]

Reinterprets bits in an unsigned short integer as a half.

**Parameters**

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

**Returns**

half

\(\ldots\)
- The reinterpreted value.

**Description**

Reinterprets the bits in the unsigned short integer \(i\) as a half-precision floating point number.
1.1.6. Half Math Functions

Half Precision Intrinsics

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

```c
__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`.

```c
__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 \texttt{half} reciprocal in round-to-nearest-even mode.

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

Returns
\begin{itemize}
\item \texttt{half} - The reciprocal of \texttt{a}.
\end{itemize}

Description
Calculates \texttt{half} reciprocal of input \texttt{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
\begin{itemize}
\item \texttt{h} - \texttt{half}. Is only being read.
\end{itemize}

Returns
\begin{itemize}
\item \texttt{half} - The nearest integer to \texttt{h}.
\end{itemize}

Description
Round \texttt{h} to the nearest integer value in half-precision floating point format, with halfway cases rounded to the nearest even integer value.

__device__ __half hrsqrt (const __half a)
Calculates \texttt{half} reciprocal square root in round-to-nearest-even mode.

Parameters
\begin{itemize}
\item \texttt{a} - \texttt{half}. Is only being read.
\end{itemize}
Returns
half
- The reciprocal square root of a.

Description
Calculates half reciprocal square root of input a in round-to-nearest mode.

```cuda
__device__ __half hsin(const __half a)
```
Calculates half sine in round-to-nearest-even mode.

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

Returns
half
- The sine of a.

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

```cuda
__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.

1.1.7. 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
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 \( \text{half2} \) vector natural logarithm in round-to-nearest-even mode.

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

Returns
\( \text{half2} \)

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

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

__device__ __half2 h2log10 (const __half2 a)
Calculates \( \text{half2} \) vector decimal logarithm in round-to-nearest-even mode.

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

Returns
\( \text{half2} \)

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

Description
Calculates \( \text{half2} \) decimal logarithm of input vector \( a \) in round-to-nearest-even mode.
```
__device__ __half2 h2log2 (const __half2 a)
Calculates half2 vector binary logarithm in round-to-nearest-even mode.

Parameters
a
  - half2. Is only being read.

Returns
half2
  - The
elementwise binary logarithm on vector a.

Description
Calculates half2 binary logarithm of input vector a in round-to-nearest mode.

__device__ __half2 h2rcp (const __half2 a)
Calculates half2 vector reciprocal in round-to-nearest-even mode.

Parameters
a
  - half2. Is only being read.

Returns
half2
  - The
elementwise reciprocal on vector a.

Description
Calculates half2 reciprocal of input vector a in round-to-nearest-even mode.

__device__ __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.

\texttt{__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.

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

Returns
\( \text{half2} \)
- 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
\( h \)
- \( \text{half2} \). Is only being read.

Returns
\( \text{half2} \)
- The truncated \( h \).

Description
Round each component of vector \( h \) to the nearest integer value that does not exceed \( h \) in magnitude.
1.2. Bfloat16 Precision Intrinsics

This section describes `nv_bfloat16` precision intrinsic functions that are only supported in device code. To use these functions include the header file `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

1.2.1. Bfloat16 Arithmetic Functions

Bfloat16 Precision Intrinsics

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

```c
__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.
\texttt{\_\_device\_\_nv\_bfloat16 \_\_habs (const \_\_nv\_bfloat16 a)}
Calculates the absolute value of input \texttt{nv\_bfloat16} number and returns the result.

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

**Returns**
\texttt{nv\_bfloat16}
- The absolute value of \texttt{a}.

**Description**
Calculates the absolute value of input \texttt{nv\_bfloat16} number and returns the result.

\texttt{\_\_device\_\_nv\_bfloat16 \_\_hadd (const \_\_nv\_bfloat16 a, const \_\_nv\_bfloat16 b)}
Performs \texttt{nv\_bfloat16} addition in round-to-nearest-even mode.

**Description**
Performs \texttt{nv\_bfloat16} addition of inputs \texttt{a} and \texttt{b}, in round-to-nearest-even mode.

\texttt{\_\_device\_\_nv\_bfloat16 \_\_hadd\_sat (const \_\_nv\_bfloat16 a, const \_\_nv\_bfloat16 b)}
Performs \texttt{nv\_bfloat16} addition 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 sum of \texttt{a} and \texttt{b}, with respect to saturation.
__device__ __nv_bfloat16 __hdiv (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs __nv_bfloat16 division in round-to-nearest-even mode.

__device__ __nv_bfloat16 __hfma (const __nv_bfloat16 a, const __nv_bfloat16 b, const __nv_bfloat16 c)
Performs __nv_bfloat16 fused multiply-add in round-to-nearest-even mode.

__device__ __nv_bfloat16 __hfma_relu (const __nv_bfloat16 a, const __nv_bfloat16 b, const __nv_bfloat16 c)
Performs __nv_bfloat16 fused multiply-add in round-to-nearest-even mode with relu saturation.

Parameters

a - __nv_bfloat16. Is only being read.
b - __nv_bfloat16. Is only being read.
c - __nv_bfloat16. Is only being read.

Returns

__nv_bfloat16

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

Performs `nv_bfloat16` multiply on inputs `a` and `b`, then performs a `nv_bfloat16` 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__ __nv_bfloat16 __hfma_sat (const __nv_bfloat16 a, const __nv_bfloat16 b, const __nv_bfloat16 c)
```

Performs `nv_bfloat16` fused multiply-add 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.
- `c` - `nv_bfloat16`. Is only being read.

**Returns**

`nv_bfloat16`

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

**Description**

Performs `nv_bfloat16` multiplication in round-to-nearest-even mode.

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

Performs `nv_bfloat16` multiplication in round-to-nearest-even mode.

**Description**

Performs `nv_bfloat16` multiplication of inputs `a` and `b`, in round-to-nearest mode.
__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

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

\textbf{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_sat (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs nv_bfloat16 subtraction 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 result of subtraction of b from a, with respect to saturation.

Description
Subtracts nv_bfloat16 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.

1.2.2. Bfloat162 Arithmetic Functions
Bfloat16 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 \texttt{nv\_bf\_float162} number and returns the result.

\texttt{__device\_\_nv\_bf\_float162\_\_hadd2 (const \_\_nv\_bf\_float162 a, const \_\_nv\_bf\_float162 b)}
Performs \texttt{nv\_bf\_float162} vector addition in round-to-nearest-even mode.

Description
Performs \texttt{nv\_bf\_float162} vector add of inputs \texttt{a} and \texttt{b}, in round-to-nearest mode.

\texttt{__device\_\_nv\_bf\_float162\_\_hadd2\_sat (const \_\_nv\_bf\_float162 a, const \_\_nv\_bf\_float162 b)}
Performs \texttt{nv\_bf\_float162} vector addition in round-to-nearest-even mode, with saturation to \([0.0, 1.0]\).

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

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

Returns
\texttt{nv\_bf\_float162}

\begin{itemize}
  \item The sum of \texttt{a} and \texttt{b}, with respect to saturation.
\end{itemize}

Description
Performs \texttt{nv\_bf\_float162} vector add of inputs \texttt{a} and \texttt{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.
**Module**

**CUDA Math API**

**version**

99

---

**b**

- `nv_bfloat162`. Is only being read.

**c**

- `nv_bfloat162`. Is only being read.

**Returns**

`nv_bfloat162`

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

**Description**

Performs `nv_bfloat162` vector multiply on inputs `a` and `b`, then performs a `nv_bfloat162` vector add of the result with `c`, rounding the result once in round-to-nearest-even mode. Then negative result is clamped to 0. NaN result is converted to canonical NaN.

```c
__device__ __nv_bfloat162 __hfma2_sat (const __nv_bfloat162 a, const __nv_bfloat162 b, const __nv_bfloat162 c)
```

Performs `nv_bfloat162` vector fused multiply-add in round-to-nearest-even mode, with saturation to $[0.0, 1.0]$.

**Parameters**

**a**

- `nv_bfloat162`. Is only being read.

**b**

- `nv_bfloat162`. Is only being read.

**c**

- `nv_bfloat162`. Is only being read.

**Returns**

`nv_bfloat162`

- The result of elementwise fused multiply-add operation on vectors `a`, `b`, and `c`, with respect to saturation.
Description
Performs \( \text{nv\_bfloat162} \) vector multiply on inputs \( a \) and \( b \), then performs a \( \text{nv\_bfloat162} \) 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.

\[
\text{__device__ __nv\_bfloat162 __hmul2 (const __nv\_bfloat162 a, const __nv\_bfloat162 b)}
\]
Performs \( \text{nv\_bfloat162} \) vector multiplication in round-to-nearest-even mode.

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

\[
\text{__device__ __nv\_bfloat162 __hmul2\_sat (const __nv\_bfloat162 a, const __nv\_bfloat162 b)}
\]
Performs \( \text{nv\_bfloat162} \) vector multiplication in round-to-nearest-even mode, with saturation to \([0.0, 1.0]\).

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

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

Returns
\( \text{nv\_bfloat162} \)
- The result of elementwise multiplication of vectors \( a \) and \( b \), with respect to saturation.

Description
Performs \( \text{nv\_bfloat162} \) vector multiplication of inputs \( a \) and \( b \), in round-to-nearest-even mode, and clamps the results to range \([0.0, 1.0]\). NaN results are flushed to +0.0.
__device__ __nv_bfloat162 __hneg2 (const __nv_bfloat162 a)
Negates both halves of the input \texttt{nv\_bfloa162} number and returns the result.

\textbf{Description}
Negates both halves of the input \texttt{nv\_bfloa162} number \(a\) and returns the result.

__device__ __nv_bfloat162 __hsub2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs \texttt{nv\_bfloa162} vector subtraction in round-to-nearest-even mode.

\textbf{Description}
Subtracts \texttt{nv\_bfloa162} input vector \(b\) from input vector \(a\) in round-to-nearest-even mode.

__device__ __nv_bfloat162 __hsub2_sat (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs \texttt{nv\_bfloa162} vector subtraction in round-to-nearest-even mode, with saturation to \([0.0, 1.0]\).

\textbf{Parameters}
- \(a\) - \texttt{nv\_bfloa162}. Is only being read.
- \(b\) - \texttt{nv\_bfloa162}. Is only being read.

\textbf{Returns}
\texttt{nv\_bfloa162}

\begin{itemize}
  \item The subtraction of vector \(b\) from \(a\), with respect to saturation.
\end{itemize}

\textbf{Description}
Subtracts \texttt{nv\_bfloa162} 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.

\subsection{Bfloat16 Comparison Functions}

Bfloat16 Precision Intrinsics
To use these functions include the header file \texttt{cuda\_bf16\_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 nv_bfloat16 greater-equal comparison.

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

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

Description
Performs nv_bfloat16 greater-equal comparison of inputs a and b. NaN inputs generate false results.

__device__ bool __hgeu (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs nv_bfloat16 unordered greater-equal comparison.

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

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

Description
Performs nv_bfloat16 greater-equal comparison of inputs a and b. NaN inputs generate true results.
__device__ bool __hgt (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs \texttt{nv\_bfloa}t16 greater-than comparison.

**Parameters**

\begin{itemize}
\item \texttt{a}
  - \texttt{nv\_bfloa}t16. Is only being read.
\item \texttt{b}
  - \texttt{nv\_bfloa}t16. Is only being read.
\end{itemize}

**Returns**

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

**Description**

Performs \texttt{nv\_bfloa}t16 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\_bfloa}t16 unordered greater-than comparison.

**Parameters**

\begin{itemize}
\item \texttt{a}
  - \texttt{nv\_bfloa}t16. Is only being read.
\item \texttt{b}
  - \texttt{nv\_bfloa}t16. Is only being read.
\end{itemize}

**Returns**

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

**Description**

Performs \texttt{nv\_bfloa}t16 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 \texttt{nv\_bf\_float16} number is infinite.

Parameters
\texttt{a}
- \texttt{nv\_bf\_float16}. Is only being read.

Returns
int
- 1
  iff \texttt{a} is equal to negative infinity,
- 0
  otherwise.

Description
Checks if the input \texttt{nv\_bf\_float16} number \texttt{a} is infinite.

__device__ bool __hisnan (const __nv_bfloat16 a)
Determine whether \texttt{nv\_bf\_float16} argument is a NaN.

Parameters
\texttt{a}
- \texttt{nv\_bf\_float16}. Is only being read.

Returns
bool
- true
  iff argument is NaN.

Description
Determine whether \texttt{nv\_bf\_float16} value \texttt{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 nv_bfloat16 less-equal comparison of inputs a and b. NaN inputs generate false results.

__device__ bool __hleu (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs nv_bfloat16 unordered less-equal comparison.

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

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

Description
Performs nv_bfloat16 less-equal comparison of inputs a and b. NaN inputs generate true results.
__device__ bool __hlt (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs __nv_bfloat16 less-than comparison.

Parameters
a
- __nv_bfloat16. Is only being read.
b
- __nv_bfloat16. Is only being read.

Returns
bool

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

Description
Performs __nv_bfloat16 less-than comparison of inputs a and b. NaN inputs generate false results.

__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

```c
__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`

**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`

**Description**

Performs `nv_bfloat16` not-equal comparison of inputs `a` and `b`. NaN inputs generate true results.
1.2.4. **Bfloat162 Comparison Functions**

**Bfloat16 Precision Intrinsics**

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

```c
__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.

```c
__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 \( \text{nv\_bfloat16} \) results of unordered if-equal comparison of vectors \( a \) and \( b \) are true;
- **false** otherwise.

**Description**

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

__device__ bool __hbge2 (const __nv_bfloat162 a, const __nv_bfloat162 b)

Performs \( \text{nv\_bfloat16} \) vector greater-equal comparison, and returns boolean true iff both \( \text{nv\_bfloat16} \) results are true, boolean false otherwise.

**Parameters**

- \( a \)
  - \( \text{nv\_bfloat16} \). Is only being read.
- \( b \)
  - \( \text{nv\_bfloat16} \). Is only being read.

**Returns**

bool

- **true** if both \( \text{nv\_bfloat16} \) results of greater-equal comparison of vectors \( a \) and \( b \) are true;
- **false** otherwise.

**Description**

Performs \( \text{nv\_bfloat16} \) vector greater-equal comparison of inputs \( a \) and \( b \). The bool result is set to true only if both \( \text{nv\_bfloat16} \) greater-equal comparisons evaluate to true, or false otherwise. NaN inputs generate false results.
__device__ bool __hbgeu2 (const __nv_bfloat162 a, const __nv_bfloat162 b)

Performs __nv_bfloat162__ vector unordered greater-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 greater-equal comparison of vectors `a` and `b` are true;
  - `false` otherwise.

**Description**

Performs __nv_bfloat162__ vector greater-equal comparison of inputs `a` and `b`. The bool result is set to true only if both __nv_bfloat16__ greater-equal comparisons evaluate to true, or false otherwise. NaN inputs generate true results.

__device__ bool __hbgt2 (const __nv_bfloat162 a, const __nv_bfloat162 b)

Performs __nv_bfloat162__ vector greater-than comparison, and returns boolean true iff both __nv_bfloat16__ results are true, boolean false otherwise.

**Parameters**

- `a` - __nv_bfloat162__. Is only being read.
- `b` - __nv_bfloat162__. Is only being read.

**Returns**

- `bool`
  - `true` if both __nv_bfloat16__ results of greater-than comparison of vectors `a` and `b` are true;
Description
Perform $\text{nv}_\text{bfloat16}$ 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 $\text{nv}_\text{bfloat16}$ vector unordered 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 $\text{nv}_\text{bfloat16}$ results of unordered greater-than comparison of vectors $a$ and $b$ are true;
- false otherwise.

Description
Perform $\text{nv}_\text{bfloat162}$ 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 true results.

__device__ bool __hble2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs $\text{nv}_\text{bfloat16}$ vector less-equal 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.
- nv_bfloat162. Is only being read.

**Returns**

bool

- trueif both nv_bfloat16 results of less-equal comparison of vectors a and b are true;
- falseotherwise.

**Description**

Performs nv_bfloat162 vector less-equal comparison of inputs a and b. The bool result is set to true only if both nv_bfloat16 less-equal comparisons evaluate to true, or false otherwise. NaN inputs generate false results.

___device___ bool __hbleu2 (const __nv_bfloat162 a, const __nv_bfloat162 b)

Performs nv_bfloat162 vector unordered less-equal comparison, and returns boolean true iff both nv_bfloat16 results are true, boolean false otherwise.

**Parameters**

- a
  - nv_bfloat162. Is only being read.
- b
  - nv_bfloat162. Is only being read.

**Returns**

bool

- trueif both nv_bfloat16 results of unordered less-equal comparison of vectors a and b are true;
- falseotherwise.

**Description**

Performs nv_bfloat162 vector less-equal comparison of inputs a and b. The bool result is set to true only if both nv_bfloat16 less-equal comparisons evaluate to true, or false otherwise. NaN inputs generate true results.
__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.

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

Description
Performs \texttt{nv\_bfloat162} 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.

__device__ bool __hbltu2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs \texttt{nv\_bfloat162} vector unordered 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.

Returns
bool
- \texttt{true} if both \texttt{nv\_bfloat16} results of unordered less-than comparison of vectors \texttt{a} and \texttt{b} are true;
Description
Performs \texttt{nv\_bfloat162} vector less-than comparison of inputs \( a \) and \( b \). 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 \texttt{nv\_bfloat162} vector not-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
- \texttt{true} if both \texttt{nv\_bfloat16} results of not-equal comparison of vectors \( a \) and \( b \) are true,
- \texttt{false} otherwise.

Description
Performs \texttt{nv\_bfloat162} vector not-equal comparison of inputs \( a \) and \( b \). The bool result is set to true only if both \texttt{nv\_bfloat16} not-equal comparisons evaluate to true, or false otherwise. NaN inputs generate false results.

\__device\__ bool \__hbneu2\ (const \__nv\_bfloat162 \( a \), const \__nv\_bfloat162 \( b \))
Performs \texttt{nv\_bfloat162} vector unordered not-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
- 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.

```c
__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__ __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__ __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.

```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__ __nv_bfloat162 __hgt2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
```

Performs `nv_bfloat162` vector greater-than comparison.

**Parameters**

- `a` - `nv_bfloat162`. Is only being read.
- `b` - `nv_bfloat162`. Is only being read.

**Returns**

- `nv_bfloat162` - The
vector result of greater-than comparison of vectors \(a\) and \(b\).

**Description**

Performs \texttt{nv\_bf\_162} vector greater-than comparison of inputs \(a\) and \(b\). The corresponding \texttt{nv\_bf\_16} results are set to 1.0 for true, or 0.0 for false. NaN inputs generate false results.

```c
__device__ __nv\_bf\_162 __hgtu2 (const __nv\_bf\_162 a, const __nv\_bf\_162 b)
```

Performs \texttt{nv\_bf\_162} vector unordered greater-than comparison.

**Parameters**

- \(a\) - \texttt{nv\_bf\_162}. Is only being read.
- \(b\) - \texttt{nv\_bf\_162}. Is only being read.

**Returns**

\texttt{nv\_bf\_162}

- The \texttt{nv\_bf\_162} vector result of unordered greater-than comparison of vectors \(a\) and \(b\).

**Description**

Performs \texttt{nv\_bf\_162} vector greater-than comparison of inputs \(a\) and \(b\). The corresponding \texttt{nv\_bf\_16} results are set to 1.0 for true, or 0.0 for false. NaN inputs generate true results.

```c
__device__ __nv\_bf\_162 __hisnan2 (const __nv\_bf\_162 a)
```

Determine whether \texttt{nv\_bf\_162} argument is a NaN.

**Parameters**

- \(a\) - \texttt{nv\_bf\_162}. Is only being read.

**Returns**

\texttt{nv\_bf\_162}

- The \texttt{nv\_bf\_162} vector result of unordered greater-than comparison of vectors \(a\) and \(b\).
**Description**

Determine whether each `nv_bfloat16` of input `nv_bfloat162` number `a` is a NaN.

```
__device__ __nv_bfloat162 __hle2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
```

Performs `nv_bfloat162` vector less-equal comparison.

**Parameters**

- **a**
  - `nv_bfloat162`. Is only being read.
- **b**
  - `nv_bfloat162`. Is only being read.

**Returns**

```
nv_bfloat162
```

The `nv_bfloat162` result of less-equal comparison of vectors `a` and `b`.

**Description**

Performs `nv_bfloat162` vector less-equal comparison of inputs `a` and `b`. The corresponding `nv_bfloat16` results are set to 1.0 for true, or 0.0 for false. NaN inputs generate false results.

```
__device__ __nv_bfloat162 __hleu2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
```

Performs `nv_bfloat162` vector unordered less-equal comparison.

**Parameters**

- **a**
  - `nv_bfloat162`. Is only being read.
- **b**
  - `nv_bfloat162`. Is only being read.

**Returns**

```
nv_bfloat162
```
The vector result of unordered less-equal comparison of vectors a and b.

**Description**

Performs `nv_bfloat162` vector less-equal comparison of inputs a and b. The corresponding `nv_bfloat16` results are set to 1.0 for true, or 0.0 for false. NaN inputs generate true results.

```
__device__ __nv_bfloat162 __hlt2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
```

Performs `nv_bfloat162` vector less-than comparison.

**Parameters**

- **a**
  - `nv_bfloat162`. Is only being read.
- **b**
  - `nv_bfloat162`. Is only being read.

**Returns**

`nv_bfloat162`

The `nv_bfloat162` vector result of less-than comparison of vectors a and b.

**Description**

Performs `nv_bfloat162` vector less-than comparison of inputs a and b. The corresponding `nv_bfloat16` results are set to 1.0 for true, or 0.0 for false. NaN inputs generate false results.

```
__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_bfloat162 vector less-than comparison of inputs a and b. The corresponding nv_bfloat16 results are set to 1.0 for true, or 0.0 for false. NaN inputs generate true results.

```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

```device__ __nv_bfloat162 __hmax2_nan (const __nv_bfloat162 a, const __nv_bfloat162 b)
```
Calculates nv_bfloat162 vector maximum of two inputs, NaNs pass through.

Description
Calculates nv_bfloat162 vector max(a, b) Elementwise nv_bfloat16 operation is defined as (a > b) ? a : b.
- If either of inputs is NaN, then canonical NaN is returned.
- If values of both inputs are 0.0, then +0.0 > -0.0
__device__ __nv_bfloat162 __hmin2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
Calculates \texttt{nv\_bfloat162} vector minimum of two inputs.

\textbf{Description}
Calculates \texttt{nv\_bfloat162} vector min(a, b) Elementwise \texttt{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 \(+0.0 > -0.0\)

__device__ __nv_bfloat162 __hmin2_nan (const __nv_bfloat162 a, const __nv_bfloat162 b)
Calculates \texttt{nv\_bfloat162} vector minimum of two inputs, NaNs pass through.

\textbf{Description}
Calculates \texttt{nv\_bfloat162} vector min(a, b) Elementwise \texttt{nv\_bfloat16} operation is defined as \((a < b) \ ? \ a : b)\.

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

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

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

\textbf{Returns}
\texttt{nv\_bfloat162}

- The vector result of not-equal comparison of vectors \texttt{a} and \texttt{b}.
Description
Performs `nv_bfloat162` vector not-equal comparison of inputs `a` and `b`. The corresponding `nv_bfloat16` results are set to 1.0 for true, or 0.0 for false. NaN inputs generate false results.

```
__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.

1.2.5. 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 __bf162__float2 __bf1622float2 (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

‣ 

\p

a converted to float2.

Description

Converts both halves of \texttt{nv\_bf\_float162} input \texttt{a} to float2 and returns the result.

\texttt{\_device\_\_nv\_bf\_float162 \_bf\_float162bf\_float162 (const \_nv\_bf\_float16 a)}

Returns \texttt{nv\_bf\_float162} with both halves equal to the input \texttt{a}.

Parameters

\texttt{a}
- \texttt{nv\_bf\_float16}. Is only being read.

Returns

\texttt{nv\_bf\_float162}

‣ 

The

vector which has both its halves equal to the input \texttt{a}.

Description

Returns \texttt{nv\_bf\_float162} number with both halves equal to the input \texttt{a} \texttt{nv\_bf\_float16} number.

\texttt{\_host\_\_device\_\_float \_bf\_float162float (const \_nv\_bf\_float16 a)}

Converts \texttt{nv\_bf\_float16} number to float.

Parameters

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

Returns

float

‣ 

\p
a converted to float.

**Description**

Converts nv_bfloat16 number a to float.

```c
__device__ int __bfloat162int_rd (__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-precision floating point value h to a signed integer in round-down mode.

```c
__device__ int __bfloat162int_rn (__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-precision floating point value h to a signed integer in round-to-nearest-even mode.
__device__ int __bfloat162int_ru (__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-precision floating point value h to a signed integer in round-up mode.

__host____device__ int __bfloat162int_rz (__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-precision floating point value h to a signed integer in round-towards-zero mode.

__device__ long long int __bfloat162ll_rd (__nv_bfloat16 h)
Convert a nv_bfloat16 to a signed 64-bit integer in round-down mode.

Parameters
h
- nv_bfloat16. Is only being read.
**Returns**
long long int

- h converted to a signed 64-bit integer.

**Description**
Convert the `nv_bfloat16`-precision floating point value `h` to a signed 64-bit integer in round-down mode.

```device__ long long int __bfloat162ll_rn (__nv_bfloat16 h)
```
Convert a `nv_bfloat16` to a signed 64-bit integer in round-to-nearest-even mode.

**Parameters**

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

**Returns**
long long int

- h converted to a signed 64-bit integer.

**Description**
Convert the `nv_bfloat16`-precision floating point value `h` to a signed 64-bit integer in round-to-nearest-even mode.

```device__ long long int __bfloat162ll_ru (__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-precision floating point value \( h \) to a signed 64-bit integer in round-up mode.

\[
\textbf{\_host\_\_device\_\ long long int \_bfloat162ll\_rz (\_nv\_bfloat16 h)}
\]
Convert a \text{nv\_bfloat16} to a signed 64-bit integer in round-towards-zero mode.

**Parameters**
\( h \)
- \text{nv\_bfloat16}. Is only being read.

**Returns**
long long int

\[
\text{\( h \) converted to a signed 64-bit integer.}
\]

Description
Convert the nv_bfloat16-precision floating point value \( h \) to a signed 64-bit integer in round-towards-zero mode.

\[
\textbf{\_device\_\ short int \_bfloat162short\_rd (\_nv\_bfloat16 h)}
\]
Convert a \text{nv\_bfloat16} to a signed short integer in round-down mode.

**Parameters**
\( h \)
- \text{nv\_bfloat16}. Is only being read.

**Returns**
short int

\[
\text{\( h \) converted to a signed short integer.}
\]

Description
Convert the nv_bfloat16-precision floating point value \( h \) to a signed short integer in round-down mode.
__device__ short int __bfloat162short_rn (__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-precision floating point value \( h \) to a signed short integer in round-to-nearest-even mode.

__device__ short int __bfloat162short_ru (__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-precision floating point value \( h \) to a signed short integer in round-up mode.
__host__ __device__ short int __bfloat162short_rz ((__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-precision floating point value h to a signed short integer in round-towards-zero mode.

__device__ unsigned int __bfloat162uint_rd ((__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-precision floating point value h to an unsigned integer in round-down mode.
__device__ unsigned int __bfloat162uint_rn (__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

Description
Convert the nv_bfloat16-precision floating point value h to an unsigned integer in round-to-nearest-even mode.

__device__ unsigned int __bfloat162uint_ru (__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

Description
Convert the nv_bfloat16-precision floating point value h to an unsigned integer in round-up mode.
__host__ __device__ unsigned int __bfloat162uint_rz(__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-precision floating point value h to an unsigned integer in round-towards-zero mode.

__device__ unsigned long long int __bfloat162ull_rd(__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-precision floating point value h to an unsigned 64-bit integer in round-down mode.
__device__ unsigned long long int __bfloat162ull_rn(__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

\hl converted to an unsigned 64-bit integer.

Description
Convert the nv_bfloat16-precision floating point value h to an unsigned 64-bit integer in round-to-nearest-even mode.

__device__ unsigned long long int __bfloat162ull_ru(__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

\hl converted to an unsigned 64-bit integer.

Description
Convert the nv_bfloat16-precision floating point value h to an unsigned 64-bit integer in round-up mode.
__host__ __device__ unsigned long long int __bfloat162ull_rz(__nv_bfloat16 h)
Convert a nv_bfloat16 to an unsigned 64-bit integer in round-towards-zero mode.

Parameters
h
   - nv_bfloat16. Is only being read.

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

Description
Convert the nv_bfloat16-precision floating point value h to an unsigned 64-bit integer in round-towards-zero mode.

__device__ unsigned short int __bfloat162ushort_rd(__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-precision floating point value h to an unsigned short integer in round-down mode.
__device__ unsigned short int __bfloat162ushort_rn(__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

Description
Convert the nv_bfloat16-precision floating point value h to an unsigned short integer in round-to-nearest-even mode.

__device__ unsigned short int __bfloat162ushort_ru(__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

Description
Convert the nv_bfloat16-precision floating point value h to an unsigned short integer in round-up mode.
__host__ __device__ unsigned short int __bfloat162ushort_rz (__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

\[
\text{h converted to an unsigned short integer.}
\]

Description

Convert the nv_bfloat16-precision floating point value \( h \) to an unsigned short integer in round-towards-zero mode.

__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

\[
\text{The reinterpreted value.}
\]

Description

Reinterprets the bits in the nv_bfloat16-precision 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-precision 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_bfloat16 __float2bfloat162_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_bfloat16 __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_rdn (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
- The 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)
Converting float number to \texttt{nv\_bfloat16} precision in round-towards-zero mode and returns \texttt{nv\_bfloat16} with converted value.

**Parameters**

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

**Returns**

\texttt{nv\_bfloat16}

- The \texttt{nv\_bfloat16} value with corresponding halves equal to the converted input floats.

**Description**

Converting float number \texttt{a} to \texttt{nv\_bfloat16} precision in round-towards-zero mode.

__host__ __device__ __nv_bfloat162 __floats2bfloat162_rn (const float a, const float b)

Converting both input floats to \texttt{nv\_bfloat16} precision in round-to-nearest-even mode and returns \texttt{nv\_bfloat162} with converted values.

**Parameters**

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

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

**Returns**

\texttt{nv\_bfloat162}

- The \texttt{nv\_bfloat162} value with corresponding halves equal to the converted input floats.

**Description**

Converting both input floats to \texttt{nv\_bfloat16} precision in round-to-nearest-even mode and combines the results into one \texttt{nv\_bfloat162} number. Low 16 bits of the return value correspond to the input \texttt{a}, high 16 bits correspond to the input \texttt{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_bfloat16 __high2bfloat16 (const __nv_bfloat162 a)
Returns high 16 bits of __nv_bfloat162 input.

Parameters
a
- __nv_bfloat162. Is only being read.

Returns
__nv_bfloat16
▷ The high 16 bits of the input.

Description
Returns high 16 bits of __nv_bfloat162 input a.
__device__ __nv_bfloat162 __high2bfloat162 (const __nv_bfloat162 a)
Extracts high 16 bits from nv_bfloat162 input.

Parameters
a
  - nv_bfloat162. Is only being read.

Returns
nv_bfloat162
  - The nv_bfloat162 with both halves equal to the high 16 bits of the input.

Description
Extracts high 16 bits from nv_bfloat162 input a and returns a new nv_bfloat162 number which has both halves equal to the extracted bits.

__host__ __device__ float __high2float (const __nv_bfloat162 a)
Converts high 16 bits of nv_bfloat162 to float and returns the result.

Parameters
a
  - nv_bfloat162. Is only being read.

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

Description
Converts high 16 bits of nv_bfloat162 input a to 32 bit floating point number and returns the result.
__device__ __nv_bfloat16 __highs2bfloat162 (const __nv_bfloat162 a, const __nv_bfloat162 b)

Extracts high 16 bits from each of the two nv_bfloat162 inputs and combines into one nv_bfloat162 number.

Parameters

a
- nv_bfloat162. Is only being read.

b
- nv_bfloat162. Is only being read.

Returns

nv_bfloat162

- The high 16 bits of a and of b.

Description

Extracts high 16 bits from each of the two nv_bfloat162 inputs and combines into one nv_bfloat162 number. High 16 bits from input a is stored in low 16 bits of the return value, high 16 bits from input b is stored in high 16 bits of the return value.

__device__ __nv_bfloat16 __int2bfloat16_rd (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-precision floating point value in round-down mode.
__host__ __device__ __nv_bfloat16 __int2bfloat16_rn (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-precision floating point value in round-to-nearest-even mode.

__device__ __nv_bfloat16 __int2bfloat16_ru (int i)
Convert a signed integer to a nv_bfloat16 in round-up 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-precision floating point value in round-up mode.

__device__ __nv_bfloat16 __int2bfloat16_rz (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

\[ i \] converted to nv_bfloat16.

Description
Convert the signed integer value \( i \) to a nv_bfloat16-precision floating point value in round-towards-zero mode.

\[
\text{__device__ \ __nv_bfloat16 \ __ldca (const \ __nv_bfloat16 \ *ptr)}
\]
Generates a \`ld.global.ca` load instruction.

Parameters
\( ptr \)
- memory location

Returns
The value pointed by `ptr`

\[
\text{__device__ \ __nv_bfloat162 \ __ldca (const \ __nv_bfloat162 \ *ptr)}
\]
Generates a \`ld.global.ca` load instruction.

Parameters
\( ptr \)
- memory location

Returns
The value pointed by `ptr`

\[
\text{__device__ \ __nv_bfloat16 \ __ldcg (const \ __nv_bfloat16 \ *ptr)}
\]
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`

__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`

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

Parameters
ptr
- memory location
__device__ __nv_bfloat162 __ldcv (const __nv_bfloat162 *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_bfloat16 __ldlu (const __nv_bfloat16 *ptr)
Generates a `ld.global.lu` load instruction.

Parameters
ptr
- memory location

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

```c
__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`

```c
__device__ __nv_bfloat16 __ll2bfloat16_rd (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

Description
Convert the signed 64-bit integer value `i` to a nv_bfloat16-precision floating point value in round-down mode.

```c
__host_device__ __nv_bfloat16 __ll2bfloa16 rn (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 \( \text{nv}_\text{bfloat16} \)-precision floating point value in round-to-nearest-even mode.

\texttt{__device__ __nv\_bfloat16 \_ll2bfloa16\_ru (long long int \( i \))}

Convert a signed 64-bit integer to a \( \text{nv}_\text{bfloat16} \) 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 \( \text{nv}_\text{bfloat16} \)-precision floating point value in round-up mode.

\texttt{__device__ __nv\_bfloat16 \_ll2bfloa16\_rz (long long int \( i \))}

Convert a signed 64-bit integer to a \( \text{nv}_\text{bfloat16} \) 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 \( \text{nv\_bfloat16} \)-precision floating point value in round-towards-zero mode.

\[
\text{__device__ __nv\_bfloat16 __low2bfloat16 (const __nv\_bfloat162 a)}
\]

Returns low 16 bits of \( \text{nv\_bfloat162} \) input.

Parameters

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

Returns

\( \text{nv\_bfloat16} \)

- Returns \( \text{nv\_bfloat16} \) which contains low 16 bits of the input \( a \).

Description

Returns low 16 bits of \( \text{nv\_bfloat162} \) input \( a \).

\[
\text{__device__ __nv\_bfloat162 __low2bfloat162 (const __nv\_bfloat162 a)}
\]

Extracts low 16 bits from \( \text{nv\_bfloat162} \) input.

Parameters

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

Returns

\( \text{nv\_bfloat162} \)

- The \( \text{nv\_bfloat162} \) with both halves equal to the low 16 bits of the input.

Description

Extracts low 16 bits from \( \text{nv\_bfloat162} \) input \( a \) and returns a new \( \text{nv\_bfloat162} \) number which has both halves equal to the extracted bits.
__host____device__ float __low2float (const __nv_bfloat162 a)
Converts low 16 bits of \texttt{nv\_bf\_float162} to float and returns the result.

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

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

Description
Converts low 16 bits of \texttt{nv\_bf\_float162} 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 \texttt{nv\_bf\_float162} input.

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

Returns
\texttt{nv\_bf\_float162}
- \texttt{a} with its halves being swapped.

Description
Swaps both halves of the \texttt{nv\_bf\_float162} input and returns a new \texttt{nv\_bf\_float162} 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 __nv_bfloat162 inputs and combines into one __nv_bfloat162 number.

Parameters

a
- __nv_bfloat162. Is only being read.

b
- __nv_bfloat162. Is only being read.

Returns

__nv_bfloat162

- The low 16 bits of a and of b.

Description

Extracts low 16 bits from each of the two __nv_bfloat162 inputs and combines into one __nv_bfloat162 number. Low 16 bits from input a is stored in low 16 bits of the return value, low 16 bits from input b is stored in high 16 bits of the return value.

__device__ __nv_bfloat16 __shfl_down_sync (unsigned mask, __nv_bfloat16 var, unsigned int delta, int width)
Exchange a variable between threads within a warp. Copy from a thread with higher ID relative to the caller.

Parameters

mask
- unsigned int. Is only being read.

var
- __nv_bfloat16. Is only being read.

delta
- int. Is only being read.

width
- int. Is only being read.
**Returns**

Returns the 2-byte word referenced by var from the source thread ID as nv_bfloat16. If the source thread ID is out of range or the source thread has exited, the calling thread’s own var is returned.

**Description**

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

```c
__device__ __nv_bfloat162 __shfl_down_sync (unsigned mask, __nv_bfloat162 var, unsigned int delta, 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.
`__device__ __nv_bfloat16 __shfl_sync (unsigned mask, __nv_bfloat16 var, int delta, 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 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__ __nv_bfloat162 __shfl_sync (unsigned mask, __nv_bfloat162 var, int delta, 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_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_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. in 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.

```c
__device__ __nv_bfloat16 __shfl_up_sync (unsigned mask, __nv_bfloat16 var, unsigned int delta, 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.

```c
__device__ __nv_bfloat16 __shfl_up_sync (unsigned mask, __nv_bfloat16 var, unsigned int delta, 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 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 subtracting delta from the caller’s lane ID. The value of var held by the resulting lane ID is returned: in effect, var is shifted up the warp by delta threads. If width is less than warpSize then each subsection of the warp behaves as a separate entity with a starting logical thread ID of 0. The source thread index will not wrap around the value of width, so effectively the lower delta threads will be unchanged. width must have a value which is a power of 2; results are undefined if width is not a power of 2, or is a number greater than warpSize.

```c
__device__ __nv_bfloat16 __shfl_xor_sync (unsigned mask, __nv_bfloat16 var, int delta, 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.
__device__ __nv_bfloat162 __shfl_xor_sync (unsigned mask, __nv_bfloat162 var, int delta, int width)

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

Parameters

- **mask**
  - unsigned int. Is only being read.
- **var**
  - __nv_bfloat162. Is only being read.
- **delta**
  - int. Is only being read.
- **width**
  - int. Is only being read.

Returns

Returns the 4-byte word referenced by var from the source thread ID as __nv_bfloat162. If the source thread ID is out of range or the source thread has exited, the calling thread’s own var is returned.
Description
Calculates a source thread ID by performing a bitwise XOR of the caller’s thread ID with mask: the value of var held by the resulting thread ID is returned. If width is less than warpSize then each group of width consecutive threads are able to access elements from earlier groups of threads, however if they attempt to access elements from later groups of threads their own value of var will be returned. This mode implements a butterfly addressing pattern such as is used in tree reduction and broadcast.

__device__ __nv_bfloat16 __short2bfloat16_rd (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-precision floating point value in round-down mode.

__host____device__ __nv_bfloat16 __short2bfloat16_rn (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 \texttt{nv\_bfloat16}-precision floating point value in round-to-nearest-even mode.

\texttt{\_\_device\_\_ \_nv\_\_bfloat16 \_\_short2bfloat16\_ru \(\text{short int} \ i\)}
Convert a signed short integer to a \texttt{nv\_bfloat16} in round-up mode.

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

Returns
\texttt{nv\_bfloat16}

\texttt{i converted to nv\_bfloat16}.

Description
Convert the signed short integer value $i$ to a \texttt{nv\_bfloat16}-precision floating point value in round-up mode.

\texttt{\_\_device\_\_ \_nv\_\_bfloat16 \_\_short2bfloat16\_rz \(\text{short int} \ i\)}
Convert a signed short integer to a \texttt{nv\_bfloat16} in round-towards-zero mode.

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

Returns
\texttt{nv\_bfloat16}

\texttt{i converted to nv\_bfloat16}.

Description
Convert the signed short integer value $i$ to a \texttt{nv\_bfloat16}-precision floating point value in round-towards-zero mode.
\__device\__ \__nv\_bf\_float16 \__short\_as\_bf\_float16 (const short int i)

Reinterprets bits in a signed short integer as a \nv\_bf\_float16.

**Parameters**

i
- short int. Is only being read.

**Returns**

\nv\_bf\_float16

- The reinterpreted value.

**Description**

Reinterprets the bits in the signed short integer \( i \) as a \nv\_bf\_float16-precision floating point number.

\__device\__ void \__st\_cg (\__nv\_bf\_float16 *ptr, \__nv\_bf\_float16 value)

Generates a `st.global.cg` store instruction.

**Parameters**

\*ptr
- memory location

value
- the value to be stored

\__device\__ void \__st\_cg (\__nv\_bf\_float162 *ptr, \__nv\_bf\_float162 value)

Generates a `st.global.cg` store instruction.

**Parameters**

\*ptr
- memory location

value
- the value to be stored
__device__ void __stcs (__nv_bfloat16 *ptr, __nv_bfloat16 value)
Generates a `st.global.cs` store instruction.

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

__device__ void __stcs (__nv_bfloat162 *ptr, __nv_bfloat162 value)
Generates a `st.global.cs` store instruction.

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

__device__ void __stwb (__nv_bfloat16 *ptr, __nv_bfloat16 value)
Generates a `st.global.wb` store instruction.

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

__device__ void __stwb (__nv_bfloat162 *ptr, __nv_bfloat162 value)
Generates a `st.global.wb` store instruction.

Parameters
ptr
- memory location
value
- the value to be stored
__device__ void __stwt (__nv_bfloat16 *ptr, __nv_bfloat16 value)
Generates a `st.global.wt` store instruction.

Parameters

ptr  
- memory location

value  
- the value to be stored

__device__ void __stwt (__nv_bfloat162 *ptr, __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 (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-precision floating point value in round-down mode.
__host__ __device__ __nv_bfloat16 __uint2bfloat16_rn (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

Description
Convert the unsigned integer value \( i \) to a nv_bfloat16-precision floating point value in round-to-nearest-even mode.

__device__ __nv_bfloat16 __uint2bfloat16_ru (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

Description
Convert the unsigned integer value \( i \) to a nv_bfloat16-precision floating point value in round-up mode.
__device__ __nv_bfloat16 __uint2bfloat16_rz (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

Description
Convert the unsigned integer value \( i \) to a nv_bfloat16-precision floating point value in round-towards-zero mode.

__device__ __nv_bfloat16 __ull2bfloat16_rd (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

Description
Convert the unsigned 64-bit integer value \( i \) to a nv_bfloat16-precision floating point value in round-down mode.
__host__ __device__ __nv_bfloat16 __ull2bfloat16_rn (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

Description

Convert the unsigned 64-bit integer value \( i \) to a nv_bfloat16-precision floating point value in round-to-nearest-even mode.

__device__ __nv_bfloat16 __ull2bfloat16_ru (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

Description

Convert the unsigned 64-bit integer value \( i \) to a nv_bfloat16-precision floating point value in round-up mode.
__device__ __nv_bfloat16 __ull2bfloat16_rz (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

Description
Convert the unsigned 64-bit integer value \( i \) to a nv_bfloat16-precision floating point value in round-towards-zero mode.

__device__ __nv_bfloat16 __ushort2bfloat16_rd (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

Description
Convert the unsigned short integer value \( i \) to a nv_bfloat16-precision floating point value in round-down mode.
__host__ __device__ __nv_bfloat16 __ ushort2bf loat16_rn (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-precision floating point value in round-to-nearest-even mode.

__device__ __nv_bfloat16 __ushort2bf loat16_ru (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-precision floating point value in round-up mode.
__device__ __nv_bfloat16 __ushort2bfloat16_rz (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

Description
Convert the unsigned short integer value \( i \) to a nv_bfloat16-precision 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

Description
Reinterprets the bits in the unsigned short integer \( i \) as a nv_bfloat16-precision floating point number.

1.2.6. Bfloat16 Math Functions
Bfloat16 Precision Intrinsics
To use these functions include the header file `cuda_bf16.h` in your program.

```c
__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`.

```c
__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.

```c
__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 $\text{nv\_bfloat16}$ natural exponential function of input a in round-to-nearest-even mode.

```c
__device__ __nv_bfloat16 hexp10 (const __nv_bfloat16 a)
```
Calculates $\text{nv\_bfloat16}$ decimal exponential function in round-to-nearest mode.

**Parameters**

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

**Returns**
nv_bfloat16

- The decimal exponential function on a.

**Description**
Calculates $\text{nv\_bfloat16}$ decimal exponential function of input a in round-to-nearest-even mode.

```c
__device__ __nv_bfloat16 hexp2 (const __nv_bfloat16 a)
```
Calculates $\text{nv\_bfloat16}$ binary exponential function in round-to-nearest mode.

**Parameters**

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

**Returns**
nv_bfloat16

- The binary exponential function on a.
Description
Calculates \texttt{nv\_bfloat16} binary exponential function of input \texttt{a} in round-to-nearest-even mode.

\texttt{\_\_device\_\_ \_nv\_bfloat16 hfloor (const \_\_nv\_bfloat16 h)}
Calculate the largest integer less than or equal to \texttt{h}.

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

Returns
\texttt{nv\_bfloat16}
- The largest integer value which is less than or equal to \texttt{h}.

Description
Calculate the largest integer value which is less than or equal to \texttt{h}.

\texttt{\_\_device\_\_ \_nv\_bfloat16 hlog (const \_\_nv\_bfloat16 a)}
Calculates \texttt{nv\_bfloat16} natural logarithm in round-to-nearest-even mode.

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

Returns
\texttt{nv\_bfloat16}
- The natural logarithm of \texttt{a}.

Description
Calculates \texttt{nv\_bfloat16} natural logarithm of input \texttt{a} in round-to-nearest-even mode.
__device__ __nv_bfloat16 hlog10 (const __nv_bfloat16 a)
Calculates \texttt{nv\_bfloa16} decimal logarithm in round-to-nearest-even mode.

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

Returns
\texttt{nv\_bfloa16}
- The decimal logarithm of \texttt{a}.

Description
Calculates \texttt{nv\_bfloa16} decimal logarithm of input \texttt{a} in round-to-nearest-even mode.

__device__ __nv_bfloat16 hlog2 (const __nv_bfloat16 a)
Calculates \texttt{nv\_bfloa16} binary logarithm in round-to-nearest-even mode.

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

Returns
\texttt{nv\_bfloa16}
- The binary logarithm of \texttt{a}.

Description
Calculates \texttt{nv\_bfloa16} binary logarithm of input \texttt{a} in round-to-nearest-even mode.

__device__ __nv_bfloat16 hrcp (const __nv_bfloat16 a)
Calculates \texttt{nv\_bfloa16} reciprocal in round-to-nearest-even mode.

Parameters
a
- \texttt{nv\_bfloa16}. Is only being read.
Returns
nv_bfloat16

- The reciprocal of a.

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

__device__ __nv_bfloat16 hrint (const __nv_bfloat16 h)
Round input to nearest integer value in nv_bfloat16-precision floating point number.

Parameters
h
- nv_bfloat16. Is only being read.

Returns
nv_bfloat16

- The nearest integer to h.

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

__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 \texttt{nv\_bf\_float16} reciprocal square root of input \texttt{a} in round-to-nearest mode.

\texttt{__device__ \_\_nv\_bf\_float16 hsin (const \_\_nv\_bf\_float16 a)}
Calculates \texttt{nv\_bf\_float16} sine in round-to-nearest-even mode.

Parameters
\texttt{a}
- \texttt{nv\_bf\_float16}. Is only being read.

Returns
\texttt{nv\_bf\_float16}
- The sine of \texttt{a}.

Description
Calculates \texttt{nv\_bf\_float16} sine of input \texttt{a} in round-to-nearest-even mode.

\texttt{__device__ \_\_nv\_bf\_float16 hsqrt (const \_\_nv\_bf\_float16 a)}
Calculates \texttt{nv\_bf\_float16} square root in round-to-nearest-even mode.

Parameters
\texttt{a}
- \texttt{nv\_bf\_float16}. Is only being read.

Returns
\texttt{nv\_bf\_float16}
- The square root of \texttt{a}.

Description
Calculates \texttt{nv\_bf\_float16} square root of input \texttt{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.

1.2.7. Bfloat16 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 nv_bfloat162 vector ceiling of the input argument.

Parameters
h
- nv_bfloat162. Is only being read.

Returns
nv_bfloat162

- The vector of smallest integers not less than h.

Description
For each component of vector h compute the smallest integer value not less than h.
__device__ __nv_bfloat162 h2cos (const __nv_bfloat162 a)
Calculates `nv_bfloat162` vector cosine in round-to-nearest-even mode.

**Parameters**

- `a` - `nv_bfloat162`. Is only being read.

**Returns**

- `nv_bfloat162`

**Description**

Calculates `nv_bfloat162` cosine of input vector `a` in round-to-nearest-even mode.

__device__ __nv_bfloat162 h2exp (const __nv_bfloat162 a)
Calculates `nv_bfloat162` vector exponential function in round-to-nearest mode.

**Parameters**

- `a` - `nv_bfloat162`. Is only being read.

**Returns**

- `nv_bfloat162`

**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 \( \text{nv\_bfloat162} \) decimal exponential function of input vector a in round-to-nearest-even mode.

```c
__device__ __nv_bfloat162 h2exp2 (const __nv_bfloat162 a)
```

Calculates \( \text{nv\_bfloat162} \) vector binary exponential function in round-to-nearest-even mode.

**Parameters**

\( a \)

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

**Returns**

nv_bfloat162

- The elementwise binary exponential function on vector a.

**Description**

Calculates \( \text{nv\_bfloat162} \) binary exponential function of input vector a in round-to-nearest-even mode.

```c
__device__ __nv_bfloat162 h2floor (const __nv_bfloat162 h)
```

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

**Parameters**

\( h \)

- \( \text{nv\_bfloat162} \). 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 \( h \).

\[
\text{__device__ __nv_bfloat162 h2log (const __nv_bfloat162 a)}
\]
Calculates \( \text{nv\_bfloa}\text{t162} \) vector natural logarithm in round-to-nearest-even mode.

Parameters
\( a \)
- \( \text{nv\_bfloa}\text{t162} \). Is only being read.

Returns
\( \text{nv\_bfloa}\text{t162} \)
- The elementwise natural logarithm on vector \( a \).

Description
Calculates \( \text{nv\_bfloa}\text{t162} \) natural logarithm of input vector \( a \) in round-to-nearest-even mode.

\[
\text{__device__ __nv_bfloat162 h2log10 (const __nv_bfloat162 a)}
\]
Calculates \( \text{nv\_bfloa}\text{t162} \) vector decimal logarithm in round-to-nearest-even mode.

Parameters
\( a \)
- \( \text{nv\_bfloa}\text{t162} \). Is only being read.

Returns
\( \text{nv\_bfloa}\text{t162} \)
- The elementwise decimal logarithm on vector \( a \).

Description
Calculates \( \text{nv\_bfloa}\text{t162} \) decimal logarithm of input vector \( a \) in round-to-nearest-even mode.
__device__ __nv_bfloat162 h2log2 (const __nv_bfloat162 a)
Calculates nv_bfloat162 vector binary logarithm in round-to-nearest-even mode.

Parameters
a
- nv_bfloat162. Is only being read.

Returns
nv_bfloat162

The
elementwise binary logarithm on vector a.

Description
Calculates nv_bfloat162 binary logarithm of input vector a in round-to-nearest mode.

__device__ __nv_bfloat162 h2rcp (const __nv_bfloat162 a)
Calculates nv_bfloat162 vector reciprocal in round-to-nearest-even mode.

Parameters
a
- nv_bfloat162. Is only being read.

Returns
nv_bfloat162

The
elementwise reciprocal on vector a.

Description
Calculates 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-precision 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 \( \mathbf{h} \) to the nearest integer value in nv_bfloat16-precision floating point format, with bfloat16way cases rounded to the nearest even integer value.

\[
\text{__device__ __nv_bfloat162 h2rsqrt (const __nv_bfloat162 a)}
\]

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.

\[
\text{__device__ __nv_bfloat162 h2sin (const __nv_bfloat162 a)}
\]

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 \texttt{nv\_bfloat162} sine of input vector \texttt{a} in round-to-nearest-even mode.

\texttt{\_\_device\_\_ \_nv\_bfloat162 h2sqrt (const \_\_nv\_bfloat162 a)}
Calculates \texttt{nv\_bfloat162} vector square root in round-to-nearest-even mode.

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

Returns
\texttt{nv\_bfloat162}
\begin{itemize}
\item The elementwise square root on vector \texttt{a}.
\end{itemize}

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

\texttt{\_\_device\_\_ \_nv\_bfloat162 h2trunc (const \_\_nv\_bfloat162 h)}
Truncate \texttt{nv\_bfloat162} vector input argument to the integral part.

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

Returns
\texttt{nv\_bfloat162}
\begin{itemize}
\item The truncated \texttt{h}.
\end{itemize}

Description
Round each component of vector \texttt{h} to the nearest integer value that does not exceed \texttt{h} in magnitude.
## 1.3. 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 `rhopot()`, `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:

```cpp
double log(double x);
float log(float x);
float logf(float x);
```

## 1.4. Single Precision Mathematical Functions

This section describes single precision mathematical functions. To use these functions you do not need to include any additional header files in your program.

### __device__ float acosf (float x)

Calculate the arc cosine of the input argument.

**Returns**

Result will be in radians, in the interval $[0, \pi]$ for $x$ inside $[-1, +1]$.

- `acosf(1)` returns $+0$.
- `acosf(x)` returns NaN for $x$ outside $[-1, +1]$.

**Description**

Calculate the principal value of the arc cosine of the input argument $x$.

**Note:**

For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.
__device__ float acoshf (float x)
Calculate the nonnegative arc hyperbolic cosine of the input argument.

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

**Description**
Calculate the nonnegative arc hyperbolic cosine of the input argument \(x\).

**Note:**
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

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

**Description**
Calculate the principal value of the arc sine of the input argument \(x\).

**Note:**
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.
__device__ float asinhf (float x)
Calculate the arc hyperbolic sine of the input argument.

Returns
▶ asinhf(0) returns 1.

Description
Calculate the arc hyperbolic sine of the input argument x.

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

__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 [- π , + π ].
▶ atan2f(0, 1) returns +0.

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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

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

Returns
Result will be in radians, in the interval [- π/2 , + π/2 ].
▶ atanf(0) returns +0.
Description
Calculate the principal value of the arc tangent of the input argument x.

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

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

Returns
- atanhf( ±0 ) returns ±0.
- atanhf( ±1 ) returns ±∞.
- atanhf(x) returns NaN for x outside interval [-1, 1].

Description
Calculate the arc hyperbolic tangent of the input argument x.

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

__device__ float cbrtf (float x)
Calculate the cube root of the input argument.

Returns
- cbrtf( ±0 ) returns ±0.
- cbrtf( ±∞ ) returns ±∞.

Description
Calculate the cube root of x, x^{1/3}. 

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.
__device__ float ceilf (float x)
Calculate ceiling of the input argument.

Returns
Returns \([x]\) expressed as a floating-point number.

- ceilf(±0) returns ±0.
- ceilf(±\(\infty\)) returns ±\(\infty\).

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(±\(\infty\)) returns NaN.

Description
Calculate the cosine of the input argument \(x\) (measured in radians).

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.
__device__ float coshf (float x)
Calculate the hyperbolic cosine of the input argument.

Returns
- coshf(0) returns 1.
- coshf(±∞) returns +∞.

Description
Calculate the hyperbolic cosine of the input argument x.

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

__device__ float cospif (float x)
Calculate the cosine of the input argument x × π.

Returns
- cospif(±0) returns 1.
- cospif(±∞) returns NaN.

Description
Calculate the cosine of x × π (measured in radians), where x is the input argument.

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.
__device__ float cyl_bessel_i0f (float x)
Calculate the value of the regular modified cylindrical Bessel function of order 0 for the input argument.

Returns
Returns the value of the regular modified cylindrical Bessel function of order 0.

Description
Calculate the value of the regular modified cylindrical Bessel function of order 0 for the input argument \( x, I_0(x) \).

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

__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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

__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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

**__device__ float erfcinvf (float y)**

Calculate the inverse complementary error function of the input argument.

**Returns**

- $\text{erfcinvf}(0)$ returns $+\infty$.
- $\text{erfcinvf}(2)$ returns $-\infty$.

**Description**

Calculate the inverse complementary error function of the input argument $y$, for $y$ in the interval $[0, 2]$. The inverse complementary error function find the value $x$ that satisfies the equation $y = \text{erfc}(x)$, for $0 \leq y \leq 2$, and $-\infty \leq x \leq \infty$.

**Note:**

For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

**__device__ float erfcxf (float x)**

Calculate the scaled complementary error function of the input argument.

**Returns**

- $\text{erfcxf}( -\infty )$ returns $+\infty$
- $\text{erfcxf}( +\infty )$ returns $+0$
- $\text{erfcxf}(x)$ returns $+\infty$ if the correctly calculated value is outside the single floating point range.

**Description**

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

Returns

- $\text{erff}(\pm0)$ returns $\pm0$.
- $\text{erff}(\pm\infty)$ returns $\pm1$.

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 y)
Calculate the inverse error function of the input argument.

Returns

- $\text{erfinvf}(1)$ returns $\infty$.
- $\text{erfinvf}(-1)$ returns $-\infty$.

Description

Calculate the inverse error function of the input argument $y$, for $y$ in the interval $[-1, 1]$. The inverse error function finds the value $x$ that satisfies the equation $y = \text{erf}(x)$, for $-1 \leq y \leq 1$, and $-\infty \leq x \leq \infty$. 
__device__ float exp10f (float x)
Calculate the base 10 exponential of the input argument.

Returns
Returns $10^x$.

Description
Calculate the base 10 exponential of the input argument $x$.

Note:
- For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.
- This function is affected by the --use_fast_math compiler flag. See the CUDA C++ Programming Guide, Appendix E.2, Table 8 for a complete list of functions affected.

__device__ float exp2f (float x)
Calculate the base 2 exponential of the input argument.

Returns
Returns $2^x$.

Description
Calculate the base 2 exponential of the input argument $x$.

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

__device__ float expf (float x)
Calculate the base $e$ exponential of the input argument.

Returns
Returns $e^x$. 
Description
Calculate the base $e$ exponential of the input argument $x$, $e^x$.

Note:
- For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.
- This function is affected by the --use_fast_math compiler flag. See the CUDA C++ Programming Guide, Appendix E.2, Table 8 for a complete list of functions affected.

__device__ float expm1f (float x)
Calculate the base $e$ exponential of the input argument, minus 1.

Returns
Returns $e^x - 1$.

Description
Calculate the base $e$ exponential of the input argument $x$, minus 1.

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

_two_device__ float fabsf (float x)
Calculate the absolute value of its argument.

Returns
Returns the absolute value of its argument.

- $\text{fabs}(\pm\infty)$ returns $\infty$.
- $\text{fabs}(\pm0)$ returns 0.

Description
Calculate the absolute value of the input argument $x$.

Note:
__device__ float fdimf(float x, float y)

Compute the positive difference between x and y.

**Returns**

Returns the positive difference between x and y.

- `fdimf(x, y)` returns `x - y` if `x > y`.
- `fdimf(x, y)` returns `+0` if `x \leq y`.

**Description**

Compute the positive difference between x and y. The positive difference is `x - y` when `x > y` and `+0` otherwise.

---

**Note:**

For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

---

__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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.
- This function is affected by the `--use_fast_math` compiler flag. See the CUDA C++ Programming Guide, Appendix E.2, Table 8 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.

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

Description
Calculate the largest integer value which is less than or equal to x.

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

__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.

- \(\text{fmaf}(\pm \infty, \pm 0, z)\) 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
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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.
__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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

__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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.
__device__ float fmodf (float x, float y)
Calculate the floating-point remainder of \( x / y \).

Returns

- Returns the floating point remainder of \( x / y \).
- \( \text{fmodf}(\pm 0, y) \) returns \( \pm 0 \) if \( y \) is not zero.
- \( \text{fmodf}(x, \pm \infty) \) returns \( x \) if \( x \) is finite.
- \( \text{fmodf}(x, y) \) returns NaN if \( x \) is \( \pm \infty \) or \( y \) is zero.
- If either argument is NaN, NaN is returned.

Description
Calculate the floating-point remainder of \( x / y \). The floating-point remainder of the division operation \( x / y \) calculated by this function is exactly the value \( x - n \times y \), 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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

__device__ float frexp (float x, int *nptr)
Extract mantissa and exponent of a floating-point value.

Returns
Returns the fractional component \( m \).

- \( \text{frexp}(0, \text{nptr}) \) returns 0 for the fractional component and zero for the integer component.
- \( \text{frexp}(\pm 0, \text{nptr}) \) 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
Decomposes the floating-point value \( x \) into a component \( m \) for the normalized fraction element and another term \( n \) for the exponent. The absolute value of \( m \) will be greater than or equal to
0.5 and less than 1.0 or it will be equal to 0; \( x = m \cdot 2^n \). The integer exponent \( n \) will be stored in the location to which \( nptr \) points.

**Note:**
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

```cpp
__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 \( \sqrt{x^2 + y^2} \). If the correct value would overflow, returns \( +\infty \). If the correct value would underflow, returns 0.

**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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

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

**Returns**
- If successful, returns the unbiased exponent of the argument.
- \( \text{ilogbf}(0) \) returns INT_MIN.
- \( \text{ilogbf}(\text{NaN}) \) returns INT_MIN.
- \( \text{ilogbf}(x) \) returns INT_MAX if \( x \) is \( \infty \) or the correct value is greater than INT_MAX.
- \( \text{ilogbf}(x) \) returns INT_MIN if the correct value is less than INT_MIN.
- Note: above behavior does not take into account FP_ILOGB0 nor FP_ILOGBNAN.

**Description**
Calculates the unbiased integer exponent of the input argument \( x \).
__device__ __RETURN_TYPE isfinite (float a)
Determine whether argument is finite.

Returns
- With Visual Studio 2013 host compiler: __RETURN_TYPE is ‘bool’. Returns true if and only if a is a finite value.
- With other host compilers: __RETURN_TYPE is ‘int’. Returns a nonzero value if and only if a is a finite value.

Description
Determine whether the floating-point value a is a finite value (zero, subnormal, or normal and not infinity or NaN).

__device__ __RETURN_TYPE isinf (float a)
Determine whether argument is infinite.

Returns
- With Visual Studio 2013 host compiler: __RETURN_TYPE is ‘bool’. Returns true if and only if a is an infinite value.
- With other host compilers: __RETURN_TYPE is ‘int’. Returns a nonzero value if and only if a is an infinite value.

Description
Determine whether the floating-point value a is an infinite value (positive or negative).

__device__ __RETURN_TYPE isnan (float a)
Determine whether argument is a NaN.

Returns
- With Visual Studio 2013 host compiler: __RETURN_TYPE is ‘bool’. Returns true if and only if a is a NaN value.
With other host compilers: __RETURN_TYPE is 'int'. Returns a nonzero value if and only if \( a \) is a NaN value.

**Description**

Determine whether the floating-point value \( a \) is a NaN.

**__device__ float j0f (float x)**

Calculate the value of the Bessel function of the first kind of order 0 for the input argument.

**Returns**

Returns the value of the Bessel function of the first kind of order 0.

- \( j0f(\pm\infty) \) returns +0.
- \( j0f(\text{NaN}) \) returns NaN.

**Description**

Calculate the value of the Bessel function of the first kind of order 0 for the input argument \( x \), \( J_0(x) \).

**Note:**

For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

**__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.

- \( j1f(\pm0) \) returns ±0.
- \( j1f(\pm\infty) \) returns ±0.
- \( j1f(\text{NaN}) \) returns NaN.

**Description**

Calculate the value of the Bessel function of the first kind of order 1 for the input argument \( x \), \( J_1(x) \).
__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 \( x \).

**Returns**
Returns the value of the Bessel function of the first kind of order \( n \).
- \( \text{jnf}(n, \text{NaN}) \) returns NaN.
- \( \text{jnf}(n, x) \) returns NaN for \( n < 0 \).
- \( \text{jnf}(n, +\infty) \) returns +0.

**Description**
Calculate the value of the Bessel function of the first kind of order \( n \) for the input argument \( x \), \( J_n(x) \).

__device__ float ldexpf (float x, int exp)
Calculate the value of \( x \cdot 2^{\text{exp}} \).

**Returns**
- \( \text{ldexpf}(x) \) returns \( \pm \infty \) if the correctly calculated value is outside the single floating point range.

**Description**
Calculate the value of \( x \cdot 2^{\text{exp}} \) of the input arguments \( x \) and \( \text{exp} \).
__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 the correctly calculated value is outside the single floating point range.
- lgammaf(\(x\)) returns +∞ if \(x \leq 0\) 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 \(\log \int_0^{\infty} e^{-t^{x-1}} dt\).

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

__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 result 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 result is undefined.

Note:

This function may be slower than alternate rounding methods. See \(\text{llrintf}\).

\___device___ float log10f(float x)\n
Calculate the base 10 logarithm of the input argument.

Returns

- \(\log_{10}(\pm 0)\) returns \(-\infty\).
- \(\log_{10}(1)\) returns \(+0\).
- \(\log_{10}(x)\) returns NaN for \(x < 0\).
- \(\log_{10}(\pm \infty)\) returns \(+\infty\).

Description

Calculate the base 10 logarithm of the input argument \(x\).

Note:

For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

\___device___ float log1pf(float x)\n
Calculate the value of \(\log_e(1 + x)\).

Returns

- \(\log_{1pf}(\pm 0)\) returns \(\pm 0\).
- \(\log_{1pf}(-1)\) returns \(-\infty\).
- \(\log_{1pf}(x)\) returns NaN for \(x < -1\).
- \(\log_{1pf}(\pm \infty)\) returns \(+\infty\).

Description

Calculate the value of \(\log_e(1 + x)\) of the input argument \(x\).
__device__ float log2f (float 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 \( x < 0 \).
- \( \log_2(\pm \infty) \) returns +\( \infty \).

Description
Calculate the base 2 logarithm of the input argument \( x \).

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

Returns
- \( \logbf(\pm 0) \) returns \(-\infty\).
- \( \logbf(\pm \infty) \) returns +\( \infty \).

Description
Calculate the floating point representation of the exponent of the input argument \( x \).
__device__ float logf(float x)
Calculate the natural logarithm of the input argument.

Returns

- logf(±0) returns $-\infty$.
- logf(1) returns +0.
- logf(x) returns NaN for $x < 0$.
- logf(+$\infty$) returns $+\infty$.

Description

Calculate the natural logarithm of the input argument $x$.

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

__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 result 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 result is undefined.
Note:
This function may be slower than alternate rounding methods. See \texttt{lrintf}.

\_\_\_device\_\_ float max (float \(a\), float \(b\))
Calculate the maximum value of the input float arguments.

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

Note, this is different from \texttt{std::} specification

\_\_\_device\_\_ float min (float \(a\), 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 \texttt{fminf} function.

Note, this is different from \texttt{std::} specification

\_\_\_device\_\_ float modff (float \(x\), float \(^*\text{iptr}\))
Break down the input argument into fractional and integral parts.

Returns
- \texttt{modff(} \(\pm \infty\), \texttt{iptr} \texttt{)} returns \(\pm 0\) and stores \(\pm \infty\) in the object pointed to by \texttt{iptr}.
- \texttt{modff(NaN,} \texttt{iptr}) stores a NaN in the object pointed to by \texttt{iptr} and returns a NaN.

Description
Break down the argument \(x\) into fractional and integral parts. The integral part is stored in the argument \texttt{iptr}. Fractional and integral parts are given the same sign as the argument \(x\).

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.
\__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.

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

\__device\__ float nearbyintf (float x)

Round the input argument to the nearest integer.

Returns

- nearbyintf(±0) returns ±0.
- nearbyintf(±\infty) returns ±\infty.

Description

Round argument x to an integer value in single precision floating-point format.

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

\__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 \( x \), \texttt{nextafterf()} returns the smallest representable number greater than \( x \).

\begin{center}
\begin{tabular}{|l|}
\hline
\textbf{Note:} \\
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6. \\
\hline
\end{tabular}
\end{center}

\begin{verbatim}
__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 \( \sqrt{p.x^2 + p.y^2 + p.z^2} \). If the correct value would overflow, returns \( +\infty \). If the correct value would underflow, returns 0.

Description

Calculates the length of three dimensional vector \( p \) in euclidean space without undue overflow or underflow.

\begin{center}
\begin{tabular}{|l|}
\hline
\textbf{Note:} \\
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6. \\
\hline
\end{tabular}
\end{center}

\begin{verbatim}
__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 \( \sqrt{p.x^2 + p.y^2 + p.z^2 + p.t^2} \). If the correct value would overflow, returns \( +\infty \). If the correct value would underflow, returns 0.

Description

Calculates the length of four dimensional vector \( p \) in euclidean space without undue overflow or underflow.
\end{verbatim}
**__device__ float normcdf (float y)**

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 y, \( \Phi(y) \).

**Note:**
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

**__device__ float normcdfinvf (float y)**

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 y, \( \Phi^{-1}(y) \). The function is defined for input values in the interval (0, 1).

**Note:**
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.
__device__ float normf (int dim, const float *a)
Calculate the square root of the sum of squares of any number of coordinates.

Returns

Returns the length of the vector \( \sqrt{p_1^2 + p_2^2 + \ldots + p\text{.dim}^2} \). If the correct value would overflow, returns \( +\infty \). If the correct value would underflow, returns 0.

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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

__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 \( \pm \infty \) for \( y \) an integer less than 0.
- \( \text{powf}(\pm 0, y) \) returns \( \pm 0 \) for \( y \) an odd integer greater than 0.
- \( \text{powf}(\pm 0, y) \) returns +0 for \( y > 0 \) and not and odd integer.
- \( \text{powf}(-1, \pm \infty) \) returns 1.
- \( \text{powf}(+1, y) \) returns 1 for any \( y \), even a NaN.
- \( \text{powf}(x, \pm 0) \) returns 1 for any \( x \), even a NaN.
- \( \text{powf}(x, y) \) returns a NaN for finite \( x < 0 \) and finite non-integer \( y \).
- \( \text{powf}(x, -\infty) \) returns \( +\infty \) for \( |x| < 1 \).
- \( \text{powf}(x, -\infty) \) returns +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.
- \( \text{powf}( -\infty, y) \) returns +0 for \( y < 0 \) and not an odd integer.
- \( \text{powf}( -\infty, y) \) returns \( -\infty \) for \( y \) an odd integer greater than 0.
- \( \text{powf}( -\infty, y) \) returns +\( \infty \) for \( y > 0 \) and not an odd integer.
- \( \text{powf}( +\infty, y) \) returns +0 for \( y < 0 \).
- `powf(+∞, y)` returns $+∞$ for $y > 0$.

**Description**

Calculate the value of $x$ to the power of $y$.

**Note:**

For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

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

Calculate reciprocal cube root function.

**Returns**

- `rcbrtf(±0)` returns $±∞$.
- `rcbrtf(±∞)` returns $±0$.

**Description**

Calculate reciprocal cube root function of $x$.

**Note:**

For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

```c
__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.
__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.
- remquof(±∞, y, quo) returns NaN.
- remquof(x, ±∞, quo) returns x.

**Description**

Compute a double-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 x and may not be the exact quotient but agrees with the exact quotient in the low order 3 bits.

__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 \( \frac{1}{\sqrt{x^2 + y^2}} \). If the square root would overflow, returns 0. If the square root would underflow, returns +∞.

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

Returns
Returns rounded integer value.

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 of the argument.

Returns
Returns one over the length of the 3D vector \( \frac{1}{\sqrt{p.x^2 + p.y^2 + p.z^2}} \). If the square root would overflow, returns 0. If the square root would underflow, returns \( +\infty \).

Description
Calculates one over the length of three dimension vector \( \mathbf{p} \) in euclidean space without undue overflow or underflow.

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.
\_

\_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 of the argument.

Returns

Returns one over the length of the 3D vector \[ \frac{1}{\sqrt{p.x^2 + p.y^2 + p.z^2}} \]. If the square root would overflow, returns 0. If the square root would underflow, returns \(+\infty\).

Description

Calculates one over the length of four dimension vector \( p \) in euclidean space without undue overflow or underflow.

Note:

For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

\_

\_device\_

float rnormf (int dim, const float *a)

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 \[ \frac{1}{\sqrt{p.1^2 + p.2^2 + \ldots + p.dim^2}} \]. If the square root would overflow, returns 0. If the square root would underflow, returns \(+\infty\).

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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.
__device__ float roundf (float x)
Round to nearest integer value in floating-point.

Returns
Returns rounded integer value.

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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

__device__ float scalblnf (float x, long 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__ 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.
__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, \( \text{sptr} \), and, respectively, third argument, \( \text{cptr} \).

See also:
\text{sinf()} and \text{cosf()}.

Note:
▶ For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.
▶ This function is affected by the \text{--use_fast_math} compiler flag. See the CUDA C++ Programming Guide, Appendix E.2, Table 8 for a complete list of functions affected.

__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:
\text{sinpif()} and \text{cospif()}. 
__device__ float sinf (float x)
Calculate the sine of the input argument.

Returns

- sinf(±0) returns ±0.
- sinf(±∞) 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

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

Description

Calculate the sine of \( x \times \pi \) (measured in radians), where \( x \) is the input argument.

**Note:**
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

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

Returns

Returns \( \sqrt{x} \):

- \( \text{sqrtf}(\pm 0) \) returns \( \pm 0 \).
- \( \text{sqrtf}(+\infty) \) returns \( +\infty \).
- \( \text{sqrtf}(x) \) returns NaN if \( x \) is less than 0.

Description

Calculate the nonnegative square root of \( x \), \( \sqrt{x} \).

**Note:**
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.
__device__ float tanf(float x)
Calculate the tangent of the input argument.

Returns

‣ tanf(±0) returns ±0.
‣ tanf(±∞) returns NaN.

Description

Calculate the tangent of the input argument x (measured in radians).

Note:

‣ For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.
‣ This function is affected by the --use_fast_math compiler flag. See the CUDA C++ Programming Guide, Appendix E.2, Table 8 for a complete list of functions affected.

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

Returns

‣ tanhf(±0) returns ±0.

Description

Calculate the hyperbolic tangent of the input argument x.

Note:

For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

__device__ float tgammaf(float x)
Calculate the gamma function of the input argument.

Returns

‣ tgammaf(±0) returns ±∞.


- \( \text{tgammaf}(2) \) returns +1.
- \( \text{tgammaf}(x) \) returns \( \pm \infty \) if the correctly calculated value is outside the single floating point range.
- \( \text{tgammaf}(x) \) returns NaN if \( x < 0 \) and \( x \) is an integer.
- \( \text{tgammaf}( -\infty ) \) returns NaN.
- \( \text{tgammaf}( +\infty ) \) 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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

---

**__device__ float truncf (float x)**

Truncate input argument to the integral part.

### Returns

Returns truncated integer value.

### 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 \( -\infty \).
- \( y0f(x) \) returns NaN for \( x < 0 \).
- \( y0f( +\infty ) \) returns +0.
- \( y0f(\text{NaN}) \) returns NaN.

### Description

Calculate the value of the Bessel function of the second kind of order 0 for the input argument \( x \), \( Y_0(x) \).
**__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 $-\infty$.
- y1f(x) returns NaN for x < 0.
- y1f($+\infty$) 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, $Y_1(x)$.

**__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 $-\infty$.
- ynf(n, x) returns NaN for x < 0.
- ynf(n, $+\infty$) 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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

1.5. 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.

```c
__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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

```c
__device__ double acosh (double x)
```
Calculate the nonnegative arc hyperbolic cosine of the input argument.

Returns
Result will be in the interval \([0, +\infty]\).
- acosh(1) returns 0.
- acosh(x) returns NaN for x in the interval \([-\infty, 1]\).

**Description**

Calculate the nonnegative arc hyperbolic cosine of the input argument x.

**Note:**

For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

```c
__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].

- asin(0) returns +0.
- asin(x) returns NaN for x outside [-1, +1].

**Description**

Calculate the principal value of the arc sine of the input argument x.

**Note:**

For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

```c
__device__ double asinh (double x)
```

Calculate the arc hyperbolic sine of the input argument.

**Returns**

- asinh(0) returns 1.

**Description**

Calculate the arc hyperbolic sine of the input argument x.

**Note:**

For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.
__device__ double atan (double x)
Calculate the arc tangent of the input argument.

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

- atan(0) returns +0.

Description
Calculate the principal value of the arc tangent of the input argument x.

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

__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 \([-\pi, +\pi]\).

- atan2(0, 1) returns +0.

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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.
**__device__ double atanh (double x)**

Calculate the arc hyperbolic tangent of the input argument.

**Returns**

- `atanh(±0)` returns ±0.
- `atanh(±1)` returns ±∞.
- `atanh(x)` returns NaN for x outside interval [-1, 1].

**Description**

Calculate the arc hyperbolic tangent of the input argument x.

**Note:**
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

**__device__ double cbrt (double x)**

Calculate the cube root of the input argument.

**Returns**

Returns \(x^{1/3}\).

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

**Description**

Calculate the cube root of x, \(x^{1/3}\).

**Note:**
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

**__device__ double ceil (double x)**

Calculate ceiling of the input argument.

**Returns**

Returns \([x]\) expressed as a floating-point number.
- ceil(±0) returns ±0.
- ceil(±∞) returns ±∞.

**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**
- cos(±0) returns 1.
- cos(±∞) returns NaN.

**Description**
Calculate the cosine of the input argument x (measured in radians).

**Note:**
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

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

**Returns**
- cosh(0) returns 1.
- cosh(±∞) returns ±∞.
Description
Calculate the hyperbolic cosine of the input argument $x$.

**Note:**
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

```
__device__ double cospi (double x)
```
Calculate the cosine of the input argument $x \cdot \pi$.

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

**Description**
Calculate the cosine of $x \cdot \pi$ (measured in radians), where $x$ is the input argument.

**Note:**
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

```
__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 $x$, $I_0(x)$.
__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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

__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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.
__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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

__device__ double erfcinv (double y)
Calculate the inverse complementary error function of the input argument.

Returns

- $\text{erfcinv}(0)$ returns $+\infty$.
- $\text{erfcinv}(2)$ returns $-\infty$.

Description
Calculate the inverse complementary error function of the input argument $y$, for $y$ in the interval $[0, 2]$. The inverse complementary error function find the value $x$ that satisfies the equation $y = \text{erfc}(x)$, for $0 \leq y \leq 2$, and $-\infty \leq x \leq \infty$.

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

__device__ double erfcx (double x)
Calculate the scaled complementary error function of the input argument.

Returns

- $\text{erfcx}(-\infty)$ returns $+\infty$
CUDA Math API

**erfcx**

- erfcx\(+\infty\) returns 0
- erfcx(\(x\)) returns +\(\infty\) if the correctly calculated value is outside the double floating point range.

**Description**

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

**Note:**

For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

**__device__ double erfinv (double y)**

Calculate the inverse error function of the input argument.

**Returns**

- erfinv(1) returns +\(\infty\).
- erfinv(-1) returns −\(\infty\).

**Description**

Calculate the inverse error function of the input argument \(y\), for \(y\) in the interval \([-1, 1]\). The inverse error function finds the value \(x\) that satisfies the equation \(y = \text{erf}(x)\), for \(-1 \leq y \leq 1\), and \(-\infty \leq x \leq \infty\).

**Note:**

For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

**__device__ double exp (double x)**

Calculate the base \(e\) exponential of the input argument.

**Returns**

Returns \(e^x\).

**Description**

Calculate the base \(e\) exponential of the input argument \(x\).
Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

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

Returns
Returns $10^x$.

Description
Calculate the base 10 exponential of the input argument $x$.

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

__device__ double exp2 (double x)
Calculate the base 2 exponential of the input argument.

Returns
Returns $2^x$.

Description
Calculate the base 2 exponential of the input argument $x$.

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

__device__ double expm1 (double x)
Calculate the base $e$ exponential of the input argument, minus 1.

Returns
Returns $e^x - 1$. 
Description
Calculate the base $e$ exponential of the input argument $x$, minus 1.

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

__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 $+\infty$.
‣ $\text{fabs}(0)$ returns 0.

Description
Calculate the absolute value of the input argument $x$.

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

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

Description
Calculates the largest integer value which is less than or equal to \(x\).

__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.

- \(\text{fma}(\pm \infty, \pm 0, z)\) 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
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.
\_\_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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

\_\_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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.
__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 \( x - n \times y \), where \( n \) is \( x / y \) with its fractional part truncated. The computed value will have the same sign as \( x \), and it’s magnitude will be less than the magnitude of \( y \).

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

__device__ double frexp (double x, int *nptr)
Extract mantissa and exponent of a floating-point value.

Returns

Returns the fractional component \( m \).

- \( \text{frexp}(0, \text{nptr}) \) returns 0 for the fractional component and zero for the integer component.
- \( \text{frexp}(\pm 0, \text{nptr}) \) 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 \( \text{nptr} \) points.

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

\[
\text{__device__ double hypot (double } x, \text{ double } y)\]

Calculate the square root of the sum of squares of two arguments.

Returns
Returns the length of the hypotenuse \( \sqrt{x^2 + y^2} \). If the correct value would overflow, returns \(+\infty\). If the correct value would underflow, returns 0.

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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

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

Compute the unbiased integer exponent of the argument.

Returns
- If successful, returns the unbiased exponent of the argument.
- \( \text{ilogb}(0) \) returns \( \text{INT\_MIN} \).
- \( \text{ilogb}(\text{NaN}) \) returns \( \text{INT\_MIN} \).
- \( \text{ilogb}(x) \) returns \( \text{INT\_MAX} \) if \( x \) is \( \infty \) or the correct value is greater than \( \text{INT\_MAX} \).
- \( \text{ilogb}(x) \) returns \( \text{INT\_MIN} \) if the correct value is less than \( \text{INT\_MIN} \).
- 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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

__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 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 an infinite value (positive or negative).

__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 a infinite value.
- With other host compilers: Returns a nonzero value if and only if \(a\) is a infinite value.

Description
Determine whether the floating-point value \(a\) is an infinite value (positive or negative).
Description

Determine whether the floating-point value \( a \) is a NaN.

\[
\text{\_\_device\_\_ double j0 (double \( x \))}
\]

Calculate the value of the Bessel function of the first kind of order 0 for the input argument.

Returns

Returns the value of the Bessel function of the first kind of order 0.

- \( j0(\pm \infty) \) returns +0.
- \( j0(\text{NaN}) \) returns NaN.

Description

Calculate the value of the Bessel function of the first kind of order 0 for the input argument \( x \), \( J_0(x) \).

Note:

For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

\[
\text{\_\_device\_\_ double j1 (double \( x \))}
\]

Calculate the value of the Bessel function of the first kind of order 1 for the input argument.

Returns

Returns the value of the Bessel function of the first kind of order 1.

- \( j1(\pm 0) \) returns +0.
- \( j1(\pm \infty) \) returns +0.
- \( j1(\text{NaN}) \) returns NaN.

Description

Calculate the value of the Bessel function of the first kind of order 1 for the input argument \( x \), \( J_1(x) \).

Note:

For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.
\_\_\_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 \( x \).

Returns
Returns the value of the Bessel function of the first kind of order \( n \).

- \( jn(n, \text{NaN}) \) returns NaN.
- \( jn(n, x) \) returns NaN for \( n < 0 \).
- \( jn(n, +\infty) \) returns +0.

Description
Calculate the value of the Bessel function of the first kind of order \( n \) for the input argument \( x \), \( J_n(x) \).

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

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

Returns
\( \text{ldexp}(x) \) returns \( \pm \infty \) if the correctly calculated value is outside the double floating point range.

Description
Calculate the value of \( x \cdot 2^{\text{exp}} \) of the input arguments \( x \) and \( \text{exp} \).

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.
__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 the correctly calculated value is outside the double floating point range.
- 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 \( \log_e \int_0^\infty e^{-t^x} dt \)

**Note:**
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

__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 result 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 result is undefined.

**Note:**
This function may be slower than alternate rounding methods. See [lrintf].

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

**Returns**
- $\log(\pm 0)$ returns $-\infty$.
- $\log(1)$ returns $+0$.
- $\log(x)$ returns NaN for $x < 0$.
- $\log(\pm \infty )$ returns $+\infty$

Description
Calculate the base $e$ logarithm of the input argument $x$.

**Note:**
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

__device__ double log10 (double x)
Calculate the base 10 logarithm of the input argument.

**Returns**
- $\log10(\pm 0)$ returns $-\infty$.
- $\log10(1)$ returns $+0$.
- $\log10(x)$ returns NaN for $x < 0$.
- $\log10(\pm \infty )$ returns $+\infty$

Description
Calculate the base 10 logarithm of the input argument $x$. 
**Note:**
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

#define double log1p (double x)
Calculate the value of $\log_e(1 + x)$.

Returns
- $\log_1p(\pm0)$ returns $\pm0$.
- $\log_1p(-1)$ returns $-\infty$.
- $\log_1p(x)$ returns NaN for $x < -1$.
- $\log_1p(+\infty)$ returns $+\infty$.

Description
Calculate the value of $\log_e(1 + x)$ of the input argument $x$.

**Note:**
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

#define double log2 (double x)
Calculate the base 2 logarithm of the input argument.

Returns
- $\log_2(\pm0)$ 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 $x$. 
__device__ double logb (double x)
Calculate the floating point representation of the exponent of the input argument.

Returns

- logb ±0 returns −∞
- logb ±∞ returns +∞

Description
Calculate the floating point representation of the exponent of the input argument x.

__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 result 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 result is undefined.
__device__ double max (double a, 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 (float a, 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 (double a, double b)
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 (double a, 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 (float a, 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 (double a, 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.

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.
__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.

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

__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.

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

__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 \( x \), \texttt{nextafter()} returns the smallest representable number greater than \( x \).

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

\texttt{__device\_\_ double norm (int dim, const double \*t)}
Calculate the square root of the sum of squares of any number of coordinates.

Returns
Returns the length of the dim-D vector \( \sqrt{p_1^2 + p_2^2 + \ldots + p_{\text{dim}}^2} \). If the correct value would overflow, returns \( +\infty \). If the correct value would underflow, returns 0. If two of the input arguments is 0, returns remaining argument.

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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

\texttt{__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 \( \sqrt{p.x^2 + p.y^2 + p.z^2} \). If the correct value would overflow, returns \( +\infty \). If the correct value would underflow, returns 0.
Description
Calculate the length of three dimensional vector \( \mathbf{p} \) in euclidean space without undue overflow or underflow.

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

\[
\text{__device__ double norm4d (double a, double b, double c, double d)}
\]
Calculate the square root of the sum of squares of four coordinates of the argument.

Returns
Returns the length of 4D vector \( \sqrt{a^2 + b^2 + c^2 + d^2} \). If the correct value would overflow, returns +\( \infty \). If the correct value would underflow, returns 0.

Description
Calculate the length of four dimensional vector \( \mathbf{p} \) in euclidean space without undue overflow or underflow.

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

\[
\text{__device__ double normcdf (double y)}
\]
Calculate the standard normal cumulative distribution function.

Returns
- \( \text{normcdf}(+\infty) \) returns 1
- \( \text{normcdf}(-\infty) \) returns +0

Description
Calculate the cumulative distribution function of the standard normal distribution for input argument \( y \), \( \Phi(y) \).
**__device__ double normcdfinv (double y)**

Calculate the inverse of the standard normal cumulative distribution function.

**Returns**

- `normcdfinv(0)` returns $-\infty$.
- `normcdfinv(1)` returns $+\infty$.
- `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 $y$, $\Phi^{-1}(y)$. The function is defined for input values in the interval (0, 1).

**__device__ double pow (double x, double y)**

Calculate the value of first argument to the power of second argument.

**Returns**

- `pow( \pm 0, y)` returns $\pm\infty$ for $y$ an integer less than 0.
- `pow( \pm 0, y)` returns $0$ for $y$ an odd integer greater than 0.
- `pow( \pm 0, y)` returns +0 for $y > 0$ and not and odd integer.
- `pow(-1, \pm\infty)` returns 1.
- `pow(+1, y)` returns 1 for any $y$, even a NaN.
- `pow(x, \pm 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, -\infty)` returns $+\infty$ for $|x| < 1$.
- `pow(x, -\infty)` returns +0 for $|x| > 1$.
- `pow(x, +\infty)` returns +0 for $|x| < 1$. 

Note:

For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.
• 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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

_device__ double rcbrt (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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

_device__ double remainder (double x, double y)
Compute double-precision floating-point remainder.

Returns
• remainder(x, 0) returns NaN.
• remainder(±∞, y) returns NaN.
- remainder\( [x, \pm \infty] \) returns \( x \) for finite \( x \).

**Description**
Compute double-precision floating-point remainder \( r \) of dividing \( x \) by \( y \) for nonzero \( y \). Thus \( r = x - ny \). The value \( n \) is the integer value nearest \( \frac{x}{y} \). In the case when \( |n - \frac{x}{y}| = \frac{1}{2} \), the even \( n \) value is chosen.

**Note:**
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

```c
__device__ double remquo (double x, double y, int *quo)
```

Compute double-precision floating-point remainder and part of quotient.

**Returns**
Retuns the remainder.
- \( \text{remquo}(x, 0, \text{quo}) \) returns NaN.
- \( \text{remquo}(\pm \infty, y, \text{quo}) \) returns NaN.
- \( \text{remquo}(x, \pm \infty, \text{quo}) \) returns \( x \).

**Description**
Compute a double-precision floating-point remainder in the same way as the `remainder[]` function. Argument \( \text{quo} \) returns part of quotient upon division of \( x \) by \( y \). Value \( \text{quo} \) 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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.
__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 \( \frac{1}{\sqrt{x^2 + y^2}} \). If the square root would overflow, returns 0. If the square root would underflow, returns \( +\infty \).

**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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

__device__ double rint (double x)

Round to nearest integer value in floating-point.

**Returns**

Returns rounded integer value.

**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 *t)

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 \( \frac{1}{\sqrt{p.1^2 + p.2^2 + \ldots + p.dim^2}} \). If the square root would overflow, returns 0. If the square root would underflow, returns \( +\infty \).

**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.
__device__ double rnorm3d (double a, double b, double c)

Calculate one over the square root of the sum of squares of three coordinates of the argument.

Returns

Returns one over the length of the 3D vector \( l = \sqrt{p.x^2 + p.y^2 + p.z^2} \). If the square root would overflow, returns 0. If the square root would underflow, returns +\( \infty \).

Description

Calculate one over the length of three dimensional vector \( p \) in euclidean space undue overflow or underflow.

__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 of the argument.

Returns

Returns one over the length of the 3D vector \( l = \sqrt{p.x^2 + p.y^2 + p.z^2 + p.t^2} \). If the square root would overflow, returns 0. If the square root would underflow, returns +\( \infty \).

Description

Calculate one over the length of four dimensional vector \( p \) in euclidean space undue overflow or underflow.
__device__ double round (double x)
Round to nearest integer value in floating-point.

Returns
Returns rounded integer value.

Description
Round x to the nearest integer value in floating-point format, with halfway cases rounded away from zero.

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

__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($\pm 0$) returns $\pm \infty$.
• 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:
This function may be slower than alternate rounding methods. See rint().
__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.
__device__ double sin (double x)
Calculate the sine of the input argument.

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

Description
Calculate the sine of the input argument x (measured in radians).

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

__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, sptr, and, respectively, third argument, cptr.

See also:
sin() and cos().

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.
__device__ void sincospi (double x, double *sptr, double *cptr)

Calculate the sine and cosine of the first input argument $x \pi$.

Returns

- none

Description

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

See also:

`sinpi()` and `cospi()`.

Note:

For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

__device__ double sinh (double x)

Calculate the hyperbolic sine of the input argument.

Returns

- $\sinh(\pm 0)$ returns $\pm 0$.
- $\sinh(\pm \infty)$ returns $\pm \infty$.

Description

Calculate the hyperbolic sine of the input argument $x$.

Note:

For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.
__device__ double sinpi (double x)
Calculate the sine of the input argument $x \times \pi$.

Returns

- $\sinpi(\pm0)$ returns $0$.
- $\sinpi(\pm\infty)$ returns NaN.

Description
Calculate the sine of $x \times \pi$ (measured in radians), where $x$ is the input argument.

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

__device__ double sqrt (double x)
Calculate the square root of the input argument.

Returns

Returns $\sqrt{x}$.

- $\sqrt{x}$ returns $0$.
- $\sqrt{+\infty}$ returns $+\infty$.
- $\sqrt{-}\ast$ returns NaN if $x$ is less than 0.

Description
Calculate the nonnegative square root of $x$, $\sqrt{x}$.

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.
__device__ double tan (double x)
Calculate the tangent of the input argument.

Returns
- $\tan(\pm0)$ returns $\pm0$.
- $\tan(\pm\infty)$ returns NaN.

Description
Calculate the tangent of the input argument $x$ (measured in radians).

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

__device__ double tanh (double x)
Calculate the hyperbolic tangent of the input argument.

Returns
- $\tanh(\pm0)$ returns 0.

Description
Calculate the hyperbolic tangent of the input argument $x$.

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

__device__ double tgamma (double x)
Calculate the gamma function of the input argument.

Returns
- $\text{tgamma}(\pm0)$ returns $\pm\infty$.
- $\text{tgamma}(2)$ returns +1.
- $\text{tgamma}(x)$ returns $\pm\infty$ if the correctly calculated value is outside the double floating point range.
‣ \( \text{tgamma}(x) \) returns NaN if \( x < 0 \) and \( x \) is an integer.
‣ \( \text{tgamma}(\rightarrow \infty) \) returns NaN.
‣ \( \text{tgamma}(+\infty) \) returns \(+\infty\).

**Description**

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

**Note:**

For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

**__device__ double trunc (double x)**

Truncate input argument to the integral part.

**Returns**

Returns truncated integer value.

**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.

‣ \( y_0(0) \) returns \(-\infty\).
‣ \( y_0(x) \) returns NaN for \( x < 0 \).
‣ \( y_0(+\infty) \) returns \(+0\).
‣ \( y_0(\text{NaN}) \) returns NaN.

**Description**

Calculate the value of the Bessel function of the second kind of order 0 for the input argument \( x \), \( Y_0(x) \).
__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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

__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, +\(\infty\)) 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 \(x\), \(Y_n(x)\).
1.6. 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 (long long int a, 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 (long long int a, 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 (unsigned long long int a, 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 (long long int a, unsigned long long int b)
Calculate the maximum value of the input long long int and unsigned long long int arguments.

Description
Calculate the maximum value of the arguments a and b, perform integer promotion first.

__device__ unsigned long long int max (unsigned long int a, unsigned long long int b)
Calculate the maximum value of the input unsigned long long int arguments.

Description
Calculate the maximum value of the arguments a and b.

__device__ long long int max (long long int a, long long int b)
Calculate the maximum value of the input long long int arguments.

Description
Calculate the maximum value of the arguments a and b.
__device__ unsigned long int max (unsigned long int a, long int b)
Calculate the maximum value of the input unsigned long int and long int arguments.

Description
Calculate the maximum value of the arguments a and b, perform integer promotion first.

__device__ unsigned long int max (long int a, unsigned long int b)
Calculate the maximum value of the input long int and unsigned long int arguments.

Description
Calculate the maximum value of the arguments a and b, perform integer promotion first.

__device__ unsigned long int max (unsigned long int a, unsigned long int b)
Calculate the maximum value of the input unsigned long int arguments.

Description
Calculate the maximum value of the arguments a and b.

__device__ long int max (long int a, 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 (unsigned int a, 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 (int a, 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 (unsigned int a, 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 (int a, 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 (unsigned long long int a, 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 (long long int a, 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 (unsigned long long int a, 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 (long long int a, 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 (unsigned long int a, 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 (long int a, 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 (unsigned long int a, 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 (long int a, 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 (unsigned int a, 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 (int a, 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 (unsigned int a, 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 (int a, 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 (unsigned long long int a, 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 (unsigned long long int a, 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 (unsigned int a, 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 (unsigned int a, unsigned int b)
Calculate the minimum value of the input unsigned int arguments.

Description
Calculate the minimum value of the arguments a and b.

1.7. 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 for this function see the CUDA C++ Programming Guide, Appendix E.2, Table 9.
- Input and output in the denormal range is flushed to sign preserving 0.0.

__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 for this function see the CUDA C++ Programming Guide, Appendix E.2, Table 9.
- Most input and output values around denormal range are flushed to sign preserving 0.0.

__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 for this function see the CUDA C++ Programming Guide, Appendix E.2, Table 9.
- Most input and output values around denormal range are flushed to sign preserving 0.0.

`__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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.
- 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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.
__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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.
- 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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.
- 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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

__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.

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

__device__ float __fdiv_ru (float x, float y)
Divide two floating point values in round-up mode.

Returns
Returns \( x / y \).

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

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.
__device__ float __fdiv_rz(float x, float y)
Divide two floating point values in round-towards-zero mode.

**Returns**
Returns \( x / y \).

**Description**
Divide two floating point values \( x \) by \( y \) in round-towards-zero mode.

**Note:**
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

__device__ float __fdividef(float x, float y)
Calculate the fast approximate division of the input arguments.

**Returns**
Returns \( x / y \).

- \(__fdividef(\infty, y)\) returns NaN for \(2^{126} < y < 2^{128}\).
- \(__fdividef(x, y)\) returns 0 for \(2^{126} < y < 2^{128}\) and \(x \neq \infty\).

**Description**
Calculate the fast approximate division of \( x \) by \( y \).

**Note:**
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.2, Table 9.
__device__ float __fmaf_ieee_rd (float x, float y, float z)
Compute fused multiply-add operation in round-down mode, ignore -ftz=true compiler flag.

Description
Behavior is the same as __fmaf_rd(x, y, z), the difference is in handling denormalized inputs and outputs: -ftz compiler flag has no effect.

__device__ float __fmaf_ieee_rn (float x, float y, float z)
Compute fused multiply-add operation in round-to-nearest-even mode, ignore -ftz=true compiler flag.

Description
Behavior is the same as __fmaf_rn(x, y, z), the difference is in handling denormalized inputs and outputs: -ftz compiler flag has no effect.

__device__ float __fmaf_ieee_ru (float x, float y, float z)
Compute fused multiply-add operation in round-up mode, ignore -ftz=true compiler flag.

Description
Behavior is the same as __fmaf_ru(x, y, z), the difference is in handling denormalized inputs and outputs: -ftz compiler flag has no effect.

__device__ float __fmaf_ieee_rz (float x, float y, float z)
Compute fused multiply-add operation in round-towards-zero mode, ignore -ftz=true compiler flag.

Description
Behavior is the same as __fmaf_rz(x, y, z), the difference is in handling denormalized inputs and outputs: -ftz compiler flag has no effect.
__device__ float __fmaf_rd (float x, float y, float z)
Compute \(x \times y + z\) as a single operation, in round-down mode.

Returns
Returns the rounded value of \(x \times y + z\) as a single operation.

- \(fmaf(\pm \infty, \pm 0, z)\) returns NaN.
- \(fmaf(\pm 0, \pm \infty, z)\) returns NaN.
- \(fmaf(x, y, -\infty)\) returns NaN if \(x \times y\) is an exact \(+\infty\).
- \(fmaf(x, y, +\infty)\) returns NaN if \(x \times y\) is an exact \(-\infty\).

Description
Computes the value of \(x \times y + z\) as a single ternary operation, rounding the result once in round-down (to negative infinity) mode.

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

__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.

- \(fmaf(\pm \infty, \pm 0, z)\) returns NaN.
- \(fmaf(\pm 0, \pm \infty, z)\) returns NaN.
- \(fmaf(x, y, -\infty)\) returns NaN if \(x \times y\) is an exact \(+\infty\).
- \(fmaf(x, y, +\infty)\) returns NaN if \(x \times y\) is an exact \(-\infty\).

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

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.
__device__ float __fmaf_ru (float x, float y, float 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.

- $\text{fmaf}(\pm \infty, \pm 0, z)$ 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-up (to positive infinity) mode.

---

**Note:**
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

---

__device__ float __fmaf_rz (float x, float y, float z)
Compute $x \times y + z$ as a single operation, in round-towards-zero mode.

**Returns**
Returns the rounded value of $x \times y + z$ as a single operation.

- $\text{fmaf}(\pm \infty, \pm 0, z)$ 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-towards-zero mode.

---

**Note:**
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.
__device__ float __fmul_rd (float x, float y)
Multiply two floating point values in round-down mode.

Returns
Returns \( x \times y \).

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

Note:
- For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.
- This operation will never be merged into a single multiply-add instruction.

__device__ float __fmul_rn (float x, float y)
Multiply two floating point values in round-to-nearest-even mode.

Returns
Returns \( x \times y \).

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

Note:
- For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.
- This operation will never be merged into a single multiply-add instruction.

__device__ float __fmul_ru (float x, float y)
Multiply two floating point values in round-up mode.

Returns
Returns \( x \times y \).
Description
Compute the product of \( x \) and \( y \) in round-up (to positive infinity) mode.

**Note:**
- For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.
- This operation will never be merged into a single multiply-add instruction.

```c
__device__ float __fmul_rz (float x, float y)
```
Multiply two floating point values in round-towards-zero mode.

**Returns**
Returns \( x \times y \).

Description
Compute the product of \( x \) and \( y \) in round-towards-zero mode.

**Note:**
- For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.
- This operation will never be merged into a single multiply-add instruction.

```c
__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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.
__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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

__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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

__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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

`__device__ float __frsqrt_rn (float x)`
Compute \( 1/\sqrt{x} \) in round-to-nearest-even mode.

Returns
Returns \( 1/\sqrt{x} \).

Description
Compute the reciprocal square root of \( x \) in round-to-nearest-even mode.

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

`__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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.
__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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

__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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

__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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.

__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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.
- 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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.
- 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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.
- 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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 6.
- This operation will never be merged into a single multiply-add instruction.

__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 for this function see the CUDA C++ Programming Guide, Appendix E.2, Table 9.
- Most input and output values around denormal range are flushed to sign preserving 0.0.

__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 for this function see the CUDA C++ Programming Guide, Appendix E.2, Table 9.
- Input and output in the denormal range is flushed to sign preserving 0.0.

__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 for this function see the CUDA C++ Programming Guide, Appendix E.2, Table 9.
Calculates 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 for this function see the CUDA C++ Programming Guide, Appendix E.2, Table 9.
- Most input and output values around denormal range are flushed to sign preserving 0.0.

__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, \( \text{sptr} \), and, respectively, third argument, \( \text{cptr} \).

Note:
- For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.2, Table 9.
- Denorm input/output is flushed to sign preserving 0.0.

\[
\text{__device__ float \_sinf (float \; x)}
\]

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 for this function see the CUDA C++ Programming Guide, Appendix E.2, Table 9.
- Input and output in the denormal range is flushed to sign preserving 0.0.

\[
\text{__device__ float \_tanf (float \; x)}
\]

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.
1.8. 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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.
- 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:
__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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.
- 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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.
- This operation will never be merged into a single multiply-add instruction.
__device__ double __ddiv_rd (double x, double y)
Divide two floating point values in round-down mode.

Returns
Returns x / y.

Description
Divides two floating point values x by y in round-down (to negative infinity) mode.

Note:
- For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.
- Requires compute capability >= 2.0.

__device__ double __ddiv_rn (double x, double y)
Divide two floating point values in round-to-nearest-even mode.

Returns
Returns x / y.

Description
Divides two floating point values x by y in round-to-nearest-even mode.

Note:
- For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.
- Requires compute capability >= 2.0.

__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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.
- Requires compute capability $\geq 2.0$.

__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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.
- Requires compute capability $\geq 2.0$.

__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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.
__device__ double __dmul_rn (double x, double y)
Multiply two floating point values in round-to-nearest-even mode.

Returns
Returns x * y.

Description
Multiplies two floating point values x and y in round-to-nearest-even mode.

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.
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 * y.

Description
Multiplies two floating point values x and y in round-up (to positive infinity) mode.

Note:
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.
This operation will never be merged into a single multiply-add instruction.

__device__ double __dmul_rz (double x, double y)
Multiply two floating point values in round-towards-zero mode.

Returns
Returns x * y.
Description
Multiplies two floating point values \( x \) and \( y \) in round-towards-zero mode.

Note:
- For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.
- This operation will never be merged into a single multiply-add instruction.

\[
\text{__device__ double __drcp_rd (double x)}
\]

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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.
- Requires compute capability \( \geq 2.0 \).

\[
\text{__device__ double __drcp_rn (double x)}
\]

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.
__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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.
- 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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.
- Requires compute capability $\geq 2.0$. 
__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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.
- Requires compute capability $\geq 2.0$.

---

__device__ double __dsqrt_rn (double x)
Compute $\sqrt{x}$ in round-to-nearest-even mode.

Returns
Returns $\sqrt{x}$.

Description
Compute the square root of $x$ in round-to-nearest-even mode.

---

**Note:**
- For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.
- Requires compute capability $\geq 2.0$.

---

__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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.
- Requires compute capability \( \geq 2.0 \).

\[ \texttt{__device__ double __dsqrt_rz (double x)} \]
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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.
- Requires compute capability \( \geq 2.0 \).

\[ \texttt{__device__ double __dsub_rd (double x, double y)} \]
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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.
__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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.
- 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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.
- 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 for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.
- This operation will never be merged into a single multiply-add instruction.

```c
__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.

**Note:**
For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

```c
__device__ double __fma_rn (double x, double y, double z)
```
Compute \( x \times y + z \) as a single operation in round-to-nearest-even mode.

**Returns**
Returns the rounded value of \( x \times y + z \) as a single operation.
- \( \text{fma}(\pm \infty, \pm 0, z) \) returns NaN.
- `fmaf(±0, ±∞, z)` returns NaN.
- `fmaf(x, y, -∞)` returns NaN if $x \times y$ is an exact $+\infty$
- `fmaf(x, y, +∞)` returns NaN if $x \times y$ is an exact $-∞$

### Description

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

### Note:

For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

---

`__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 $x \times y$ is an exact $+∞$
- `fmaf(x, y, +∞)` returns NaN if $x \times y$ is an exact $-∞$

### Description

Computes the value of $x \times y + z$ as a single ternary operation, rounding the result once in round-up (to positive infinity) mode.

### Note:

For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.
__device__ double __fma_rz (double x, double y, double z)

Compute $x \times y + z$ as a single operation in round-towards-zero mode.

Returns

Returns the rounded value of $x \times y + z$ as a single operation.

- fma($\pm \infty$, $\pm 0$, $z$) returns NaN.
- fma($\pm 0$, $\pm \infty$, $z$) returns NaN.
- fma($x$, $y$, $-\infty$) returns NaN if $x \times y$ is an exact $+\infty$
- 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-towards-zero mode.

Note:

For accuracy information for this function see the CUDA C++ Programming Guide, Appendix E.1, Table 7.

1.9. Integer Intrinsics

This section describes integer intrinsic functions that are only supported in device code. To use these functions you do not need to include any additional header files in your program.

__device__ unsigned int __brev (unsigned int x)

Reverse the bit order of a 32 bit unsigned integer.

Returns

Returns the bit-reversed value of $x$. i.e. bit $N$ of the return value corresponds to bit $31-N$ of $x$.

Description

Reverses the bit order of the 32 bit unsigned integer $x$. 
__device__ unsigned long long int __brevll (unsigned long long int x)
Reverse the bit order of a 64 bit unsigned integer.

Returns
Returns the bit-reversed value of x. i.e. bit N of the return value corresponds to bit 63-N of x.

Description
Reverses the bit order of the 64 bit unsigned integer x.

__device__ unsigned int __byte_perm (unsigned int x, unsigned int y, unsigned int s)
Return selected bytes from two 32 bit unsigned integers.

Returns
The returned value r is computed to be: result[n] := input[selector[n]] where result[n] is the nth byte of r.

Description
byte_perm(x,y,s) returns a 32-bit integer consisting of four bytes from eight input bytes provided in the two input integers x and y, as specified by a selector, s.


The selector indices are as follows (the upper 16-bits of the selector are not used): selector[0] = s<2:0> selector[1] = s<6:4> selector[2] = s<10:8> selector[3] = s<14:12>

__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 (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 (\(\text{shift} \& 31\)). The least significant 32-bits of the result are returned.

\[
\text{__device__ unsigned int __funnelshift_rc (unsigned int lo, unsigned int hi, unsigned int shift)}
\]

Concatenate \(hi:lo\), shift right by \(\min(\text{shift}, 32)\) 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 \(\text{shift}\). 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 \(\text{shift} \{\min(\text{shift}, 32)\}\). The least significant 32-bits of the result are returned.

\[
\text{__device__ int __hadd (int, int)}
\]

Compute average of signed input arguments, avoiding overflow in the intermediate sum.

Returns
Returns a signed integer value representing the signed average value of the two inputs.

Description
Compute average of signed input arguments \(x\) and \(y\) as \((x + y) >> 1\), avoiding overflow in the intermediate sum.

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

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 \(x \ast y\).

Description
Calculate the least significant 32 bits of the product of the least significant 24 bits of \(x\) and \(y\). The high order 8 bits of \(x\) and \(y\) are ignored.
__device__ long long int __mul64hi (long long int x, long long int y)
Calculate the most significant 64 bits of the product of the two 64 bit integers.

Returns
Returns the most significant 64 bits of the product \( x \times y \).

Description
Calculate the most significant 64 bits of the 128-bit product \( x \times y \), where \( x \) and \( y \) are 64-bit integers.

__device__ int __mulhi (int x, int y)
Calculate the most significant 32 bits of the product of the two 32 bit integers.

Returns
Returns the most significant 32 bits of the product \( x \times y \).

Description
Calculate the most significant 32 bits of the 64-bit product \( x \times y \), where \( x \) and \( y \) are 32-bit integers.

__device__ int __popc (unsigned int x)
Count the number of bits that are set to 1 in a 32 bit integer.

Returns
Returns a value between 0 and 32 inclusive representing the number of set bits.

Description
Count the number of bits that are set to 1 in \( x \).

__device__ int __popcll (unsigned long long int x)
Count the number of bits that are set to 1 in a 64 bit integer.

Returns
Returns a value between 0 and 64 inclusive representing the number of set bits.
Description
Count the number of bits that are set to 1 in x.

__device__ int __rhadd (int, int)
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 \((x + y + 1) \gg 1\), 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, unsigned int)
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.

\[\text{__device__ unsigned int __umul24 (unsigned int x, unsigned int y)}\]
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.

\[\text{__device__ unsigned long long int __umul64hi (unsigned long long int x, unsigned long long int y)}\]
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.

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

\[
\_
\_
\_
\_
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\..
__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)
Convert a double to a signed int in round-towards-zero mode.

Returns
Returns converted value.

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

__device__ long long int __double2ll_rd (double x)
Convert a double to a signed 64-bit int in round-down mode.

Returns
Returns converted value.

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

__device__ long long int __double2ll_rn (double x)
Convert a double to a signed 64-bit int in round-to-nearest-even mode.

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

\[
\text{__device__ long long int __double2ll_ru (double } x)\]
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.

\[
\text{__device__ long long int __double2ll_rz (double)}\]
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.

\[
\text{__device__ int __double2loint (double } x)\]
Reinterpret low 32 bits in a double as a signed integer.

Returns
Returns reinterpreted value.

Description
Reinterpret the low 32 bits in the double-precision floating point value \( x \) as a signed integer.
__device__ unsigned int __double2uint_rd (double x)
Convert a double to an unsigned int in round-down mode.

Returns
Returns converted value.

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

__device__ unsigned int __double2uint_rn (double x)
Convert a double to an unsigned int in round-to-nearest-even mode.

Returns
Returns converted value.

Description
Convert the double-precision floating point value x to an unsigned integer value in round-to-nearest-even mode.

__device__ unsigned int __double2uint_ru (double x)
Convert a double to an unsigned int in round-up mode.

Returns
Returns converted value.

Description
Convert the double-precision floating point value x to an unsigned integer value in round-up (to positive infinity) mode.

__device__ unsigned int __double2uint_rz (double)
Convert a double to an unsigned int in round-towards-zero mode.

Returns
Returns converted value.
Description
Convert the double-precision floating point value \( x \) to an unsigned integer value in round-towards-zero mode.

```c
__device__ unsigned long long int __double2ull_rd (double x)
```
Convert a double to an unsigned 64-bit int in round-down mode.

Returns
Returns converted value.

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

```c
__device__ unsigned long long int __double2ull_rn (double x)
```
Convert a double to an unsigned 64-bit int in round-to-nearest-even mode.

Returns
Returns converted value.

Description
Convert the double-precision floating point value \( x \) to an unsigned 64-bit integer value in round-to-nearest-even mode.

```c
__device__ unsigned long long int __double2ull_ru (double x)
```
Convert a double to an unsigned 64-bit int in round-up mode.

Returns
Returns converted value.

Description
Convert the double-precision floating point value \( x \) to an unsigned 64-bit integer value in round-up (to positive infinity) mode.
__device__ unsigned long long int __double2ull_rz (double)
Convert a double to an unsigned 64-bit int in round-towards-zero mode.

Returns
Returns converted value.

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

__device__ long long int __double_as_longlong (double x)
Reinterpret bits in a double as a 64-bit signed integer.

Returns
Returns reinterpreted value.

Description
Reinterpret the bits in the double-precision floating point value \( x \) as a signed 64-bit integer.

__device__ int __float2int_rd (float x)
Convert a float to a signed integer in round-down mode.

Returns
Returns converted value.

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

__device__ int __float2int_rn (float x)
Convert a float to a signed integer in round-to-nearest-even mode.

Returns
Returns converted value.
Description
Convert the single-precision floating point value \( x \) to a signed integer in round-to-nearest-even mode.

\[ \text{__device__ int __float2int_ru (float) } \]
Convert a float to a signed integer in round-up mode.

Returns
Returns converted value.

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

\[ \text{__device__ int __float2int_rz (float x) } \]
Convert a float to a signed integer in round-towards-zero mode.

Returns
Returns converted value.

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

\[ \text{__device__ long long int __float2ll_rd (float x) } \]
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.
__device__ long long int __float2ll_rn (float x)
Convert a float to a signed 64-bit integer in round-to-nearest-even mode.

Returns
Returns converted value.

Description
Convert the single-precision floating point value x to a signed 64-bit integer in round-to-nearest-even mode.

__device__ long long int __float2ll_ru (float x)
Convert a float to a signed 64-bit integer in round-up mode.

Returns
Returns converted value.

Description
Convert the single-precision floating point value x to a signed 64-bit integer in round-up (to positive infinity) mode.

__device__ long long int __float2ll_rz (float x)
Convert a float to a signed 64-bit integer in round-towards-zero mode.

Returns
Returns converted value.

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

__device__ unsigned int __float2uint_rd (float x)
Convert a float to an unsigned integer in round-down mode.

Returns
Returns converted value.
Description
Convert the single-precision floating point value x to an unsigned integer in round-down (to negative infinity) mode.

__device__ unsigned int __float2uint_rn (float x)
Convert a float to an unsigned integer in round-to-nearest-even mode.

Returns
Returns converted value.

Description
Convert the single-precision floating point value x to an unsigned integer in round-to-nearest-even mode.

__device__ unsigned int __float2uint_ru (float x)
Convert a float to an unsigned integer in round-up mode.

Returns
Returns converted value.

Description
Convert the single-precision floating point value x to an unsigned integer in round-up (to positive infinity) mode.

__device__ unsigned int __float2uint_rz (float x)
Convert a float to an unsigned integer in round-towards-zero mode.

Returns
Returns converted value.

Description
Convert the single-precision floating point value x to an unsigned integer in round-towards-zero mode.
__device__ unsigned long long int __float2ull_rd (float x)
Convert a float to an unsigned 64-bit integer in round-down mode.

Returns
Returns converted value.

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

__device__ unsigned long long int __float2ull_rn (float x)
Convert a float to an unsigned 64-bit integer in round-to-nearest-even mode.

Returns
Returns converted value.

Description
Convert the single-precision floating point value \( x \) to an unsigned 64-bit integer in round-to-nearest-even mode.

__device__ unsigned long long int __float2ull_ru (float x)
Convert a float to an unsigned 64-bit integer in round-up mode.

Returns
Returns converted value.

Description
Convert the single-precision floating point value \( x \) to an unsigned 64-bit integer in round-up (to positive infinity) mode.
**__device__ unsigned long long int __float2ull_rz (float x)**

Convert a float to an unsigned 64-bit integer in round-towards-zero mode.

**Returns**

Returns converted value.

**Description**

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

**__device__ int __float_as_int (float x)**

Reinterpret bits in a float as a signed integer.

**Returns**

Returns reinterpreted value.

**Description**

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

**__device__ unsigned int __float_as_uint (float x)**

Reinterpret bits in a float as an unsigned integer.

**Returns**

Returns reinterpreted value.

**Description**

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

**__device__ double __hiloint2double (int hi, int lo)**

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

**Returns**

Returns reinterpreted value.
**Description**

Reinterpret the integer value of hi as the high 32 bits of a double-precision floating point value and the integer value of lo as the low 32 bits of the same double-precision floating point value.

**__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.

**__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.

__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.

__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.

__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.

__device__ float __uint_as_float (unsigned int x)
Reinterpret bits in an unsigned integer as a float.

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

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

Returns
Returns converted value.

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

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

Returns
Returns converted value.

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

\[
\text{__device__ double __ull2double_ru (unsigned long long int x)}
\]
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.

1.11. 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. Result is stored as unsigned int and returned.
__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. Result is stored 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. Result is stored as unsigned int and returned.

__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. Result is stored as unsigned int and returned.
__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. Result is stored as unsigned int and returned.

__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. Result is stored as unsigned int and returned.

__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. Result is stored as unsigned int and returned.
__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. Result is stored as unsigned int and returned.

__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. Result is stored as unsigned int and returned.

__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. Result is stored as unsigned int and returned.
__device__ unsigned int __vaddss2 (unsigned int a, unsigned int b)
Perform 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. Result is stored as unsigned int and returned.

__device__ unsigned int __vaddss4 (unsigned int a, unsigned int b)
Perform 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. Result is stored as unsigned int and returned.

__device__ unsigned int __vaddus2 (unsigned int a, unsigned int b)
Perform 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. Result is stored as unsigned int and returned.

__device__ unsigned int __vavgs4 (unsigned int a, unsigned int b)
Computes per-byte signed rounder 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. Result is stored as unsigned int and returned.
__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. Result is stored as unsigned int and returned.

__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. Result is stored as unsigned int and returned.

__device__ unsigned int __vcmpeq2 (unsigned int a, unsigned int b)
Performs per-halfword (un)signed comparison.

Returns
Returns 0xffffffff computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts result is ffff if they are equal, and 0000 otherwise. For example __vcmpeq2(0x1234aba5, 0x1234aba6) returns 0xffffffff.
__device__ unsigned int __vcmpeq4 (unsigned int a, unsigned int b)

Performs per-byte (un)signed comparison.

Returns

Returns 0xff if a = b, else returns 0.

Description

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts result is ff if they are equal, and 00 otherwise. For example __vcmpeq4(0x1234aba5, 0x1234aba6) returns 0xffffff00.

__device__ unsigned int __vcmpges2 (unsigned int a, unsigned int b)

Performs per-halfword signed comparison: a >= b ? 0xffff : 0.

Returns

Returns 0xffff if a >= b, else returns 0.

Description

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts result is ffff if ‘a’ part >= ‘b’ part, and 0000 otherwise. For example __vcmpges2(0x1234aba5, 0x1234aba6) returns 0xffff0000.

__device__ unsigned int __vcmpges4 (unsigned int a, unsigned int b)

Performs per-byte signed comparison.

Returns

Returns 0xff if a >= b, else returns 0.

Description

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts result is ff if ‘a’ part >= ‘b’ part, and 00 otherwise. For example __vcmpges4(0x1234aba5, 0x1234aba6) returns 0xffffff00.
__device__ unsigned int __vcmpgeu2 (unsigned int a, unsigned int b)
Performs per-halfword unsigned comparison: a >= 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 ? 0xffff : 0.

Returns
Returns 0xffff if a <= b, else returns 0.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts result is ffff if ‘a’ part <= ‘b’ part, and 0000 otherwise. For example __vcmples2(0x1234aba5, 0x1234aba6) returns 0xffffffff.

__device__ unsigned int __vcmples4 (unsigned int a, unsigned int b)
Performs per-byte signed comparison.

Returns
Returns 0xff if a <= b, else returns 0.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts result is ff if ‘a’ part <= ‘b’ part, and 00 otherwise. For example __vcmples4(0x1234aba5, 0x1234aba6) returns 0xffffffff.

__device__ unsigned int __vcmpleu2 (unsigned int a, unsigned int b)
Performs per-halfword unsigned comparison: a <= b ? 0xffff : 0.

Returns
Returns 0xffff if a <= b, else returns 0.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts result is ffff if ‘a’ part <= ‘b’ part, and 0000 otherwise. For example __vcmpleu2(0x1234aba5, 0x1234aba6) returns 0xffffffff.
__device__ unsigned int __vcmpleu4 (unsigned int a, unsigned int b)
Performs per-byte unsigned comparison.

Returns
Returns 0xff if a <= 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 __vcmlts2 (unsigned int a, unsigned int b)
Performs per-halfword signed comparison: a < b ? 0xffff : 0.

Returns
Returns 0xffff if a < b, else returns 0.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts result is ffff if 'a' part < 'b' part, and 0000 otherwise. For example __vcmlts2(0x1234aba5, 0x1234aba6) returns 0x0000ffff.

__device__ unsigned int __vcmlts4 (unsigned int a, unsigned int b)
Performs per-byte signed comparison.

Returns
Returns 0xff if a < b, else returns 0.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts result is ff if 'a' part < 'b' part, and 00 otherwise. For example __vcmlts4(0x1234aba5, 0x1234aba6) returns 0x000000ff.
__device__ unsigned int __vcmpltu2 (unsigned int a, unsigned int b)
Performs per-halfword unsigned comparison: \( a < b \) ? \( 0xffff \) : 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: \( a \neq b \) ? \( 0xffff \) : 0.

Returns
Returns \( 0xffff \) if \( a \neq b \), else returns 0.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts result is \( ffff \) if ‘a’ part \( \neq \) ‘b’ part, and \( 0000 \) otherwise. For example __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. Result is stored as unsigned int and returned.

\__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. Result is stored as unsigned int and 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. Result is stored as unsigned int and returned.

__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. Result is stored as unsigned int and returned.

__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. Result is stored as unsigned int and returned.
__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. Result is stored as unsigned int and returned.

__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. Result is stored as unsigned int and returned.

__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. Result is stored as unsigned int and returned.
__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. Result is stored as unsigned int and returned.

__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. Result is stored as unsigned int and returned.

__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. Result is stored as unsigned int and returned.
__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. Result is stored as unsigned int and returned.

__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. Result is stored as unsigned int and returned.

__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. Result is stored as unsigned int and returned.

__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 functions computes absolute difference and sum it up. Result is stored as unsigned int and returned.

__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 functions computes absolute difference and sum it up. Result is stored as unsigned int and returned.

__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 af abs difference of unsigned.

Returns
Returns computed value.
Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function computes absolute differences, and returns sum of those differences.

`__device__ unsigned int __vseteq2 (unsigned int a, unsigned int b)`
Performs per-halfword (un)signed comparison.

Returns
Returns 1 if a = b, else returns 0.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function performs comparison ‘a’ part == ‘b’ part. If both equalities are satisfied, function returns 1.

`__device__ unsigned int __vseteq4 (unsigned int a, unsigned int b)`
Performs per-byte (un)signed comparison.

Returns
Returns 1 if a = b, else returns 0.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function performs comparison ‘a’ part == ‘b’ part. If both equalities are satisfied, function returns 1.

`__device__ unsigned int __vsetges2 (unsigned int a, unsigned int b)`
Performs per-halfword signed comparison.

Returns
Returns 1 if a >= b, else returns 0.
Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function performs comparison ‘a’ part >= ‘b’ part. If both inequalities are satisfied, function returns 1.

__device__ unsigned int __vsetges4 (unsigned int a, unsigned int b)
Performs per-byte signed comparison.

Returns
Returns 1 if a >= b, else returns 0.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function performs comparison ‘a’ part >= ‘b’ part. If both inequalities are satisfied, function returns 1.

__device__ unsigned int __vsetgeu2 (unsigned int a, unsigned int b)
Performs per-halfword unsigned minimum unsigned comparison.

Returns
Returns 1 if a >= b, else returns 0.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function performs comparison ‘a’ part >= ‘b’ part. If both inequalities are satisfied, function returns 1.

__device__ unsigned int __vsetgeu4 (unsigned int a, unsigned int b)
Performs per-byte unsigned comparison.

Returns
Returns 1 if a >= b, else returns 0.
Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function performs comparison ‘a’ part >= ‘b’ part. If both inequalities are satisfied, function returns 1.

__device__ unsigned int __vsetgts2 (unsigned int a, unsigned int b)
Performs per-halfword signed comparison.

Returns
Returns 1 if a > b, else returns 0.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function performs comparison ‘a’ part > ‘b’ part. If both inequalities are satisfied, function returns 1.

__device__ unsigned int __vsetgts4 (unsigned int a, unsigned int b)
Performs per-byte signed comparison.

Returns
Returns 1 if a > b, else returns 0.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function performs comparison ‘a’ part > ‘b’ part. If both inequalities are satisfied, function returns 1.

__device__ unsigned int __vsetgtu2 (unsigned int a, unsigned int b)
Performs per-halfword unsigned comparison.

Returns
Returns 1 if a > b, else returns 0.
Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function performs comparison 'a' part > 'b' part. If both inequalities are satisfied, function returns 1.

__device__ unsigned int __vsetgtu4 (unsigned int a, unsigned int b)
Performs per-byte unsigned comparison.

Returns
Returns 1 if a > b, else returns 0.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function performs comparison 'a' part > 'b' part. If both inequalities are satisfied, function returns 1.

__device__ unsigned int __vsetles2 (unsigned int a, unsigned int b)
Performs per-halfword unsigned minimum computation.

Returns
Returns 1 if a <= b, else returns 0.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function performs comparison 'a' part <= 'b' part. If both inequalities are satisfied, function returns 1.

__device__ unsigned int __vsetles4 (unsigned int a, unsigned int b)
Performs per-byte signed comparison.

Returns
Returns 1 if a <= b, else returns 0.
Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function performs comparison ‘a’ part <= ‘b’ part. If both inequalities are satisfied, function returns 1.

__device__ unsigned int __vsetleu2 (unsigned int a, unsigned int b)
Performs per-halfword signed comparison.

Returns
Returns 1 if a <= b, else returns 0.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function performs comparison ‘a’ part <= ‘b’ part. If both inequalities are satisfied, function returns 1.

__device__ unsigned int __vsetleu4 (unsigned int a, unsigned int b)
Performs per-byte unsigned comparison.

Returns
Returns 1 if a <= b, else returns 0.

Description
Splits 4 bytes of each argument into 4 part, 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 __vsetltls2 (unsigned int a, unsigned int b)
Performs per-halfword signed comparison.

Returns
Returns 1 if a < b, else returns 0.
Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function performs comparison ‘a’ part <= ‘b’ part. If both inequalities are satisfied, function returns 1.

__device__ unsigned int __vsetlts4 (unsigned int a, unsigned int b)
Performs per-byte signed comparison.

Returns
Returns 1 if a < b, else returns 0.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function performs comparison ‘a’ part <= ‘b’ part. If both inequalities are satisfied, function returns 1.

__device__ unsigned int __vsetltu2 (unsigned int a, unsigned int b)
Performs per-halfword unsigned comparison.

Returns
Returns 1 if a < b, else returns 0.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function performs comparison ‘a’ part <= ‘b’ part. If both inequalities are satisfied, function returns 1.

__device__ unsigned int __vsetltu4 (unsigned int a, unsigned int b)
Performs per-byte unsigned comparison.

Returns
Returns 1 if a < b, else returns 0.
Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function performs comparison ‘a’ part <= ‘b’ part. If both inequalities are satisfied, function returns 1.

__device__ unsigned int __vsetne2 (unsigned int a, unsigned int b)
Performs per-halfword (un)signed comparison.

Returns
Returns 1 if a != b, else returns 0.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function performs comparison ‘a’ part != ‘b’ part. If both conditions are satisfied, function returns 1.

__device__ unsigned int __vsetne4 (unsigned int a, unsigned int b)
Performs per-byte (un)signed comparison.

Returns
Returns 1 if a != b, else returns 0.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function performs comparison ‘a’ part != ‘b’ part. If both conditions are satisfied, function returns 1.

__device__ unsigned int __vsub2 (unsigned int a, unsigned int b)
Performs per-halfword (un)signed substraction, 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 functions performs substraction. Result is stored as unsigned int and returned.

__device__ unsigned int __vsub4 (unsigned int a, unsigned int b)
Performs per-byte substraction.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts functions performs substraction. Result is stored as unsigned int and returned.

__device__ unsigned int __vsubss2 (unsigned int a, unsigned int b)
Performs per-halfword (un)signed substraction, 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 functions performs substraction with signed saturation. Result is stored as unsigned int and returned.

__device__ unsigned int __vsubss4 (unsigned int a, unsigned int b)
Performs per-byte substraction 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 functions performs substraction with signed saturation. Result is stored as unsigned int and returned.

__device__ unsigned int __vsubus2 (unsigned int a, unsigned int b)
Performs per-halfword substraction 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 functions performs substraction with unsigned saturation. Result is stored as unsigned int and returned.

__device__ unsigned int __vsubus4 (unsigned int a, unsigned int b)
Performs per-byte substraction 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 functions performs substraction with unsigned saturation. Result is stored as unsigned int and returned.
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.

VESA DisplayPort

DisplayPort and DisplayPort Compliance Logo, DisplayPort Compliance Logo for Dual-mode Sources, and DisplayPort Compliance Logo for Active Cables are trademarks owned by the Video Electronics Standards Association in the United States and other countries.

HDMI

HDMI, the HDMI logo, and High-Definition Multimedia Interface are trademarks or registered trademarks of HDMI Licensing LLC.

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

© 2007-2019 NVIDIA Corporation. All rights reserved.