Table of Contents

Chapter 1. Modules .............................................................................................................. 1
  1.1. Half Precision Intrinsics ............................................................................................. 2
      Half Arithmetic Functions .............................................................................................. 2
      Half2 Arithmetic Functions ............................................................................................ 2
      Half Comparison Functions ............................................................................................ 2
      Half2 Comparison Functions .......................................................................................... 2
      Half Precision Conversion and Data Movement ............................................................ 2
      Half Math Functions ....................................................................................................... 2
      Half2 Math Functions ..................................................................................................... 2
  1.1.1. Half Arithmetic Functions ....................................................................................... 2
         __habs............................................................................................................................ 2
         __hadd.......................................................................................................................... 3
         __hadd_rn...................................................................................................................... 3
         __hadd_sat.................................................................................................................... 3
         __hddiv........................................................................................................................ 4
         __hfma........................................................................................................................ 4
         __hfma_relu.................................................................................................................. 4
         __hfma_sat................................................................................................................... 5
         __hmul.......................................................................................................................... 5
         __hmul_rn....................................................................................................................... 6
         __hmul_sat..................................................................................................................... 6
         __hneg.......................................................................................................................... 6
         __hsub.......................................................................................................................... 6
         __hsub_rn....................................................................................................................... 7
         __hsub_sat..................................................................................................................... 7
         atomicAdd....................................................................................................................... 7
  1.1.2. Half2 Arithmetic Functions ..................................................................................... 8
         __h2div............................................................................................................................ 8
         __habs2.......................................................................................................................... 8
         __hadd2.......................................................................................................................... 9
         __hadd2_rn...................................................................................................................... 9
         __hadd2_sat.................................................................................................................... 9
         __hcmmadd................................................................................................................... 10
         __hfma2........................................................................................................................ 10
         __hfma2_relu.................................................................................................................. 11
         __hfma2_sat................................................................................................................... 11
1.1.3. Half Comparison Functions

__heq.......................................................................................................................... 14
__hequ.............................................................................................................................. 15
__hge............................................................................................................................... 16
__hgeu.............................................................................................................................. 16
__hgt.................................................................................................................................... 17
__hgtu............................................................................................................................... 17
__hisinf............................................................................................................................ 18
__hisnan............................................................................................................................ 18
__hle................................................................................................................................. 19
__hleu............................................................................................................................... 19
__hlt.................................................................................................................................... 20
__hltu................................................................................................................................. 20
__hmax............................................................................................................................... 21
__hmax_nan....................................................................................................................... 21
__hmin................................................................................................................................. 21
__hmin_nan....................................................................................................................... 21
__hne.................................................................................................................................... 22
__hneu............................................................................................................................... 22

1.1.4. Half2 Comparison Functions

__hbeq2......................................................................................................................... 23
__hbequ2......................................................................................................................... 23
__hbge2............................................................................................................................ 24
__hbgeu2.......................................................................................................................... 24
__hbgt2.............................................................................................................................. 25
__hbgtu2........................................................................................................................... 26
__hble2.............................................................................................................................. 26
__hbleu2........................................................................................................................... 27
__hbleu2........................................................................................................................... 27
__hblt2.............................................................................................................................. 28
__hblt2.............................................................................................................................. 28
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>__hbne2</td>
<td>29</td>
</tr>
<tr>
<td>__hbneu2</td>
<td>29</td>
</tr>
<tr>
<td>__heq2</td>
<td>30</td>
</tr>
<tr>
<td>__hequ2</td>
<td>30</td>
</tr>
<tr>
<td>__hge2</td>
<td>31</td>
</tr>
<tr>
<td>__hgeu2</td>
<td>31</td>
</tr>
<tr>
<td>__hgt2</td>
<td>32</td>
</tr>
<tr>
<td>__htu2</td>
<td>32</td>
</tr>
<tr>
<td>__hisnan2</td>
<td>33</td>
</tr>
<tr>
<td>__hle2</td>
<td>33</td>
</tr>
<tr>
<td>__hleu2</td>
<td>34</td>
</tr>
<tr>
<td>__hlt2</td>
<td>34</td>
</tr>
<tr>
<td>__htlu2</td>
<td>34</td>
</tr>
<tr>
<td>__hmax2</td>
<td>35</td>
</tr>
<tr>
<td>__hmax2_nan</td>
<td>35</td>
</tr>
<tr>
<td>__hmin2</td>
<td>36</td>
</tr>
<tr>
<td>__hmin2_nan</td>
<td>36</td>
</tr>
<tr>
<td>__hne2</td>
<td>36</td>
</tr>
<tr>
<td>__hneu2</td>
<td>37</td>
</tr>
</tbody>
</table>

### 1.1.5. Half Precision Conversion and Data Movement

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>__double2half</td>
<td>37</td>
</tr>
<tr>
<td>__float22half2_rn</td>
<td>38</td>
</tr>
<tr>
<td>__float2half</td>
<td>38</td>
</tr>
<tr>
<td>__float2half2_rn</td>
<td>39</td>
</tr>
<tr>
<td>__float2half_rd</td>
<td>39</td>
</tr>
<tr>
<td>__float2half_rn</td>
<td>40</td>
</tr>
<tr>
<td>__float2half_ru</td>
<td>40</td>
</tr>
<tr>
<td>__float2half_rz</td>
<td>41</td>
</tr>
<tr>
<td>__floats2half2_rn</td>
<td>41</td>
</tr>
<tr>
<td>__half22float2</td>
<td>42</td>
</tr>
<tr>
<td>__half2float</td>
<td>42</td>
</tr>
<tr>
<td>__half2half2</td>
<td>42</td>
</tr>
<tr>
<td>__half2half2</td>
<td>42</td>
</tr>
<tr>
<td>__half2int_rd</td>
<td>43</td>
</tr>
<tr>
<td>__half2int_rn</td>
<td>43</td>
</tr>
<tr>
<td>__half2int_ru</td>
<td>44</td>
</tr>
<tr>
<td>__half2int_rz</td>
<td>44</td>
</tr>
<tr>
<td>__half2ll_rd</td>
<td>45</td>
</tr>
<tr>
<td>__half2ll_rn</td>
<td>45</td>
</tr>
</tbody>
</table>
1.1.6. Half Math Functions

- **hceil**
- **hcos**
- **hexp**
- **hexp2**
- **hfloor**
- **hlog**
- **hlog2**
- **hlog10**
- **hlog2**
- **hrcp**
- **hrnt**
- **hrsqrt**
- **hsin**
- **hstt**
- **htrunc**

1.1.7. Half2 Math Functions

- **h2ceil**
- **h2cos**
- **h2exp**
- **h2exp2**
- **h2floor**
- **h2log**
- **h2log10**
- **h2log2**
- **h2rcp**
- **h2rint**
- **h2rsqrt**
- **h2sin**
1.2. Bfloat16 Precision Intrinsics ................................................................. 96
1.2.1. Bfloat16 Arithmetic Functions ......................................................... 97
  __h2div ........................................................................................................ 97
  __habs ........................................................................................................... 97
  __hadd ........................................................................................................... 97
  __hadd_rn ................................................................................................... 98
  __hadd_sat ................................................................................................. 98
  __hdiv ........................................................................................................... 98
  __hfma ......................................................................................................... 99
  __hfma_relu ............................................................................................... 99
  __hfma_sat ................................................................................................. 100
  __hmul ......................................................................................................... 100
  __hmul_rn ................................................................................................. 100
  __hmul_sat ................................................................................................. 101
  __hneg ......................................................................................................... 101
  __hsub ......................................................................................................... 101
  __hsub_rn ................................................................................................. 102
  __hsub_sat ................................................................................................. 102
  atomicAdd ................................................................................................. 102
1.2.2. Bfloat162 Arithmetic Functions .................................................... 103
  __habs2 ...................................................................................................... 103
  __hadd2 ...................................................................................................... 104
  __hadd2_rn ............................................................................................... 104
  __hadd2_sat ............................................................................................. 104
  __hcmadd ................................................................................................. 105
  __hfma2 ...................................................................................................... 105
  __hfma2_relu ........................................................................................... 105
  __hfma2_sat ............................................................................................. 106
  __hmul2 ...................................................................................................... 107
__hmul2_rn............................................................................................................................ 107
__hmul2_sat.......................................................................................................................... 107
__hneg2................................................................................................................................ 108
__hsub2................................................................................................................................ 108
__hsub2_rn............................................................................................................................ 108
__hsub2_sat........................................................................................................................... 108
atomicAdd.............................................................................................................................. 109

1.2.3. Bfloat16 Comparison Functions.................................................................................... 110
__heq...................................................................................................................................... 110
__hequ.................................................................................................................................... 110
__hge.................................................................................................................................... 111
__hgeu................................................................................................................................... 111
__hgt..................................................................................................................................... 112
__hgtu.................................................................................................................................... 112
__hisinf................................................................................................................................ 113
__hisnan................................................................................................................................. 113
__hle....................................................................................................................................... 114
__hleu.................................................................................................................................... 114
__hlt....................................................................................................................................... 114
__hltu..................................................................................................................................... 115
__hmax................................................................................................................................... 116
__hmax_nan.......................................................................................................................... 116
__hmin................................................................................................................................... 116
__hmin_nan........................................................................................................................... 117
__hne...................................................................................................................................... 117
__hneu................................................................................................................................... 117

1.2.4. Bfloat162 Comparison Functions.................................................................................. 118
__hbeq2.................................................................................................................................. 118
__hbequ2................................................................................................................................ 119
__hbge2.................................................................................................................................. 119
__hbgeu2................................................................................................................................ 120
__hbgt2.................................................................................................................................. 121
__hbgtu2................................................................................................................................ 121
__hble2.................................................................................................................................. 122
__hbleu2................................................................................................................................ 123
__hblt2................................................................................................................................... 123
__hbltu2................................................................................................................................ 124
__hbne2.................................................................................................................................. 125
1.2.5. Bfloat16 Precision Conversion and Data Movement

__bfloat162float2................................................................. 134
__bfloat162bfloat162.......................................................... 135
__bfloat162float................................................................. 135
__bfloat162int_rd............................................................... 136
__bfloat162int_rn............................................................... 136
__bfloat162int_ru............................................................... 137
__bfloat162int_rz............................................................... 137
__bfloat162ll_rd................................................................. 138
__bfloat162ll_rn................................................................. 138
__bfloat162ll_ru................................................................. 139
__bfloat162ll_rz................................................................. 139
__bfloat162short_rd.......................................................... 140
__bfloat162short_rn.......................................................... 140
__bfloat162short_ru.......................................................... 141
__bfloat162short_rz.......................................................... 141
__bfloat162uint_rd........................................................... 141
__bfloat162uint_rn........................................................... 142
__bfloat162uint_ru........................................................... 143
__bfloat162uint_rz........................................................... 143
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>__ull2bfloat16_ru</td>
<td>180</td>
</tr>
<tr>
<td>__ull2bfloat16_rz</td>
<td>181</td>
</tr>
<tr>
<td>__ushort2bfloat16_rd</td>
<td>181</td>
</tr>
<tr>
<td>__ushort2bfloat16_rn</td>
<td>182</td>
</tr>
<tr>
<td>__ushort2bfloat16_ru</td>
<td>182</td>
</tr>
<tr>
<td>__ushort2bfloat16_rz</td>
<td>183</td>
</tr>
<tr>
<td>__ushort_as_bfloat16</td>
<td>183</td>
</tr>
<tr>
<td>1.2.6. Bfloat16 Math Functions</td>
<td>183</td>
</tr>
<tr>
<td>hceil</td>
<td>184</td>
</tr>
<tr>
<td>hcos</td>
<td>184</td>
</tr>
<tr>
<td>hexp</td>
<td>184</td>
</tr>
<tr>
<td>hexp10</td>
<td>185</td>
</tr>
<tr>
<td>hexp2</td>
<td>185</td>
</tr>
<tr>
<td>hfloor</td>
<td>186</td>
</tr>
<tr>
<td>hlog</td>
<td>186</td>
</tr>
<tr>
<td>hlog10</td>
<td>187</td>
</tr>
<tr>
<td>hlog2</td>
<td>187</td>
</tr>
<tr>
<td>hrscrp</td>
<td>187</td>
</tr>
<tr>
<td>hrint</td>
<td>188</td>
</tr>
<tr>
<td>hsqrt</td>
<td>188</td>
</tr>
<tr>
<td>h2ceil</td>
<td>190</td>
</tr>
<tr>
<td>h2cos</td>
<td>191</td>
</tr>
<tr>
<td>h2exp</td>
<td>191</td>
</tr>
<tr>
<td>h2exp10</td>
<td>191</td>
</tr>
<tr>
<td>h2exp2</td>
<td>192</td>
</tr>
<tr>
<td>h2floor</td>
<td>192</td>
</tr>
<tr>
<td>h2log</td>
<td>193</td>
</tr>
<tr>
<td>h2log10</td>
<td>193</td>
</tr>
<tr>
<td>h2log2</td>
<td>194</td>
</tr>
<tr>
<td>h2rcp</td>
<td>194</td>
</tr>
<tr>
<td>h2rint</td>
<td>194</td>
</tr>
<tr>
<td>h2sqrt</td>
<td>195</td>
</tr>
<tr>
<td>h2sin</td>
<td>195</td>
</tr>
<tr>
<td>h2sqrt</td>
<td>196</td>
</tr>
</tbody>
</table>
h2trunc

1.3. Mathematical Functions

1.4. Single Precision Mathematical Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>acosf</td>
<td>197</td>
</tr>
<tr>
<td>acoshf</td>
<td>198</td>
</tr>
<tr>
<td>asinf</td>
<td>198</td>
</tr>
<tr>
<td>asinhf</td>
<td>199</td>
</tr>
<tr>
<td>atan2f</td>
<td>199</td>
</tr>
<tr>
<td>atanf</td>
<td>199</td>
</tr>
<tr>
<td>atanhf</td>
<td>200</td>
</tr>
<tr>
<td>cbrtf</td>
<td>200</td>
</tr>
<tr>
<td>ceilf</td>
<td>201</td>
</tr>
<tr>
<td>copysignf</td>
<td>201</td>
</tr>
<tr>
<td>cosf</td>
<td>201</td>
</tr>
<tr>
<td>coshf</td>
<td>202</td>
</tr>
<tr>
<td>cospif</td>
<td>202</td>
</tr>
<tr>
<td>cyl_bessel_i0f</td>
<td>203</td>
</tr>
<tr>
<td>cyl_bessel_i1f</td>
<td>203</td>
</tr>
<tr>
<td>erfcf</td>
<td>203</td>
</tr>
<tr>
<td>erfcinvf</td>
<td>204</td>
</tr>
<tr>
<td>erfcxf</td>
<td>204</td>
</tr>
<tr>
<td>erff</td>
<td>205</td>
</tr>
<tr>
<td>erfinvf</td>
<td>205</td>
</tr>
<tr>
<td>exp10f</td>
<td>206</td>
</tr>
<tr>
<td>exp2f</td>
<td>206</td>
</tr>
<tr>
<td>expf</td>
<td>207</td>
</tr>
<tr>
<td>expm1f</td>
<td>207</td>
</tr>
<tr>
<td>fabsf</td>
<td>207</td>
</tr>
<tr>
<td>fdimf</td>
<td>208</td>
</tr>
<tr>
<td>fdividef</td>
<td>208</td>
</tr>
<tr>
<td>floorf</td>
<td>209</td>
</tr>
<tr>
<td>fmaf</td>
<td>209</td>
</tr>
<tr>
<td>fmaxf</td>
<td>210</td>
</tr>
<tr>
<td>fminf</td>
<td>210</td>
</tr>
<tr>
<td>fmodf</td>
<td>211</td>
</tr>
<tr>
<td>frexp</td>
<td>211</td>
</tr>
<tr>
<td>hypotf</td>
<td>212</td>
</tr>
<tr>
<td>ilogbf</td>
<td>213</td>
</tr>
</tbody>
</table>
isfinite........................................................................................................................................ 213
isinf............................................................................................................................................ 214
isnan.......................................................................................................................................... 214
j0f............................................................................................................................................... 214
j1f............................................................................................................................................... 215
jnf............................................................................................................................................... 215
ldexpf......................................................................................................................................... 216
lgammaf.................................................................................................................................... 216
llrintf.......................................................................................................................................... 217
llroundf...................................................................................................................................... 217
log10f......................................................................................................................................... 218
log1pf......................................................................................................................................... 218
log2f........................................................................................................................................... 219
logbf........................................................................................................................................... 219
logf............................................................................................................................................. 220
lrintf........................................................................................................................................... 220
lroundf....................................................................................................................................... 221
max............................................................................................................................................ 221
min............................................................................................................................................. 221
modff..........................................................................................................................................221
nanf............................................................................................................................................ 222
nearbyintf...................................................................................................................................222
nextafterf................................................................................................................................... 223
norm3df..................................................................................................................................... 223
norm4df..................................................................................................................................... 224
normcdf.................................................................................................................................... 224
normcdfinvf................................................................................................................................225
normf......................................................................................................................................... 225
powf........................................................................................................................................... 226
rcbrtf.......................................................................................................................................... 227
remainderf.................................................................................................................................227
remquoof................................................................................................................................... 228
rhypotf........................................................................................................................................ 228
rintf........................................................................................................................................... 229
rnorm3df....................................................................................................................................229
rnorm4df....................................................................................................................................229
rnormf........................................................................................................................................ 230
roundf....................................................................................................................................... 230
rsqrtf
scalarmf
scalbnf
signbit
sincosf
sincospif
sinf
sinhf
sinpif
sqrtf
tanf
tanhf
tgammaf
truncf
y0f
y1f
ynf
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..............................................................................................................................................248
exp10...........................................................................................................................................248
exp2............................................................................................................................................248
expm1..........................................................................................................................................248
fabs.............................................................................................................................................249
fdim............................................................................................................................................249
floor..........................................................................................................................................250
fma.............................................................................................................................................251
fmax............................................................................................................................................251
fmin............................................................................................................................................251
fmod............................................................................................................................................252
frexp...........................................................................................................................................252
hypot..........................................................................................................................................253
ilogb...........................................................................................................................................253
isfinite......................................................................................................................................254
isinf...........................................................................................................................................255
isnan..........................................................................................................................................255
j0................................................................................................................................................255
j1................................................................................................................................................256
jn................................................................................................................................................256
ldexp..........................................................................................................................................257
lgamma......................................................................................................................................257
llrint..........................................................................................................................................258
llround......................................................................................................................................258
log..............................................................................................................................................259
log10..........................................................................................................................................259
log1p..........................................................................................................................................260
log2............................................................................................................................................260
logb............................................................................................................................................261
lrint............................................................................................................................................261
lround......................................................................................................................................261
max.............................................................................................................................................262
max.............................................................................................................................................262
max.............................................................................................................................................262
min.............................................................................................................................................262
min.............................................................................................................................................262
min.............................................................................................................................................263
min.............................................................................................................................................263
modf..........................................................................................................................................263
nan............................................................................................................................................. 264
nearbyint.................................................................................................................................... 264
nextafter.................................................................................................................................... 264
norm.......................................................................................................................................... 265
norm3d...................................................................................................................................... 265
norm4d...................................................................................................................................... 266
normcdf..................................................................................................................................... 266
normcdfinv.................................................................................................................................267
pow.............................................................................................................................................267
rcbrt........................................................................................................................................... 268
remainder..................................................................................................................................269
remquo...................................................................................................................................... 269
rhypot.........................................................................................................................................270
rint..............................................................................................................................................270
rnorm........................................................................................................................................ 271
rnorm3d.....................................................................................................................................271
rnorm4d.....................................................................................................................................272
round..........................................................................................................................................272
rsqrt........................................................................................................................................... 272
scalbln....................................................................................................................................... 273
scalbn........................................................................................................................................ 273
signbit........................................................................................................................................ 274
sin.............................................................................................................................................. 274
sincos.........................................................................................................................................274
sincospi.......................................................................................................................................275
sinh............................................................................................................................................ 276
sinpi........................................................................................................................................... 276
sqrt.............................................................................................................................................276
tan.............................................................................................................................................. 277
tanh............................................................................................................................................277
tgamma..................................................................................................................................... 278
tgamma.................................................................................................................................... 278
trunc.......................................................................................................................................... 278
y0................................................................................................................................................ 279
y1................................................................................................................................................ 279
yn................................................................................................................................................ 280
1.6. Integer Mathematical Functions.......................................................................................... 280
abs............................................................................................................................................. 280
labs............................................................................................................................................. 281
1.7. Single Precision Intrinsics

__cosf
__exp10f
__expf
__fadd_rd
__fadd_rn
__fadd_ru
1.8. Double Precision Intrinsics

- __dadd_rd__ .................................................................................................................... 305
- __dadd_rn__ ..................................................................................................................... 306
- __dadd_ru__ ..................................................................................................................... 306
- __dadd_rz__ ..................................................................................................................... 307
- __ddiv_rd__ ..................................................................................................................... 307
- __ddiv_rn__ ..................................................................................................................... 307
- __ddiv_ru__ ..................................................................................................................... 308
- __ddiv_rz__ ..................................................................................................................... 308
- __dmul_rd__ .................................................................................................................... 309
- __dmul_rn__ .................................................................................................................... 309
- __dmul_ru__ .................................................................................................................... 309
- __dmul_rz__ .................................................................................................................... 310
- __drcp_rd__ ..................................................................................................................... 310
- __drcp_rn__ ..................................................................................................................... 311
- __drcp_ru__ ..................................................................................................................... 311
- __drcp_rz__ ..................................................................................................................... 312
- __dsqrt_rd__ .................................................................................................................... 312
- __dsqrt_rn__ .................................................................................................................... 312
- __dsqrt_ru__ .................................................................................................................... 313
- __dsqrt_rz__ .................................................................................................................... 313
- __dsub_rd__ ..................................................................................................................... 314
- __dsub_rn__ ..................................................................................................................... 314
- __dsub_ru__ ..................................................................................................................... 315
- __dsub_rz__ ..................................................................................................................... 315
- __fma_rd__ ..................................................................................................................... 315
- __fma_rn__ ..................................................................................................................... 316
- __fma_ru__ ..................................................................................................................... 317
- __fma_rz__ ..................................................................................................................... 317

1.9. Integer Intrinsics

- __brev__ ......................................................................................................................... 318
- __brevll__ ....................................................................................................................... 318
- __byte_perm__ .............................................................................................................. 318
- __clz__ ........................................................................................................................... 319
- __clzll__ ......................................................................................................................... 319
- __ffs__ ............................................................................................................................ 319
- __ffsll__ .......................................................................................................................... 320
1.10. Type Casting Intrinsics ....................................................................................................... 326
__double2float_rd..................................................................................................................... 326
__double2float_rn..................................................................................................................... 326
__double2float_ru..................................................................................................................... 326
__double2float_rz......................................................................................................................327
__double2hiint...........................................................................................................................327
__double2int_rd...................................................................................................................... 327
__double2int_rn...................................................................................................................... 328
__double2int_ru...................................................................................................................... 328
__double2int_rz.........................................................................................................................328
__double2ll_rd........................................................................................................................ 328
__double2ll_rn........................................................................................................................ 329
__double2ll_ru........................................................................................................................ 329
__double2ll_rz........................................................................................................................ 329
__double2loint.......................................................... 330
__double2uint_rd...................................................................................................................... 330
__double2uint_rn...................................................................................................................... 330
__double2uint_ru...................................................................................................................... 330
__double2uint_rz.......................................................................................................................331
__double2ull_rd........................................................................................................................ 331
1.11. SIMD Intrinsics

__vabs2
__vabs4
__vabsdiffs2
__vabssss2
__vadd2
__vadd4
__vaddss2
__vaddss4
__vaddus2
__vaddus4
__vavgs2
__vavgs4
__vavgu2
__vavgu4
__vcmp2
__vcmp4
__vcmpges2
__vcmpges4
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>__vcmpgeu4</td>
<td>354</td>
</tr>
<tr>
<td>__vcmpgtu2</td>
<td>355</td>
</tr>
<tr>
<td>__vcmpgtu4</td>
<td>355</td>
</tr>
<tr>
<td>__vcmpgtu2</td>
<td>355</td>
</tr>
<tr>
<td>__vcmpgtu4</td>
<td>356</td>
</tr>
<tr>
<td>__vcmples2</td>
<td>356</td>
</tr>
<tr>
<td>__vcmples4</td>
<td>356</td>
</tr>
<tr>
<td>__vcmpleu2</td>
<td>357</td>
</tr>
<tr>
<td>__vcmpleu4</td>
<td>357</td>
</tr>
<tr>
<td>__vcmpplts2</td>
<td>357</td>
</tr>
<tr>
<td>__vcmpplts4</td>
<td>358</td>
</tr>
<tr>
<td>__vcmppltu2</td>
<td>358</td>
</tr>
<tr>
<td>__vcmppltu4</td>
<td>358</td>
</tr>
<tr>
<td>__vcmpne2</td>
<td>359</td>
</tr>
<tr>
<td>__vcmpne4</td>
<td>359</td>
</tr>
<tr>
<td>__vhaddu2</td>
<td>359</td>
</tr>
<tr>
<td>__vhaddu4</td>
<td>360</td>
</tr>
<tr>
<td>__vmaxs2</td>
<td>360</td>
</tr>
<tr>
<td>__vmaxs4</td>
<td>360</td>
</tr>
<tr>
<td>__vmaxu2</td>
<td>361</td>
</tr>
<tr>
<td>__vmaxu4</td>
<td>361</td>
</tr>
<tr>
<td>__vmmins2</td>
<td>361</td>
</tr>
<tr>
<td>__vmmins4</td>
<td>362</td>
</tr>
<tr>
<td>__vminu2</td>
<td>362</td>
</tr>
<tr>
<td>__vminu4</td>
<td>362</td>
</tr>
<tr>
<td>__vneg2</td>
<td>363</td>
</tr>
<tr>
<td>__vneg4</td>
<td>363</td>
</tr>
<tr>
<td>__vnegss2</td>
<td>363</td>
</tr>
<tr>
<td>__vnegss4</td>
<td>363</td>
</tr>
<tr>
<td>__vsads2</td>
<td>364</td>
</tr>
<tr>
<td>__vsads4</td>
<td>364</td>
</tr>
<tr>
<td>__vsadu2</td>
<td>364</td>
</tr>
<tr>
<td>__vsadu4</td>
<td>365</td>
</tr>
<tr>
<td>__vseteq2</td>
<td>365</td>
</tr>
<tr>
<td>__vseteq4</td>
<td>365</td>
</tr>
<tr>
<td>__vsetges2</td>
<td>366</td>
</tr>
<tr>
<td>__vsetges4</td>
<td>366</td>
</tr>
<tr>
<td>__vsetgeu2</td>
<td>366</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
1.1. Half Precision Intrinsics

This section describes half precision intrinsic functions that are only supported in device code. To use these functions, include the header file `cuda_fp16.h` in your program.

Half Arithmetic Functions

Half2 Arithmetic Functions

Half Comparison Functions

Half2 Comparison Functions

Half Precision Conversion and Data Movement

Half Math Functions

Half2 Math Functions

1.1.1. Half Arithmetic Functions

Half Precision Intrinsics

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

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

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

Parameters

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

half

- The absolute value of a.

**Description**

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

```c
__device__ __half __hadd (const __half a, const __half b)
```

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

**Description**

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

```c
__device__ __half __hadd_rn (const __half a, const __half b)
```

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

**Description**

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

```c
__device__ __half __hadd_sat (const __half a, const __half b)
```

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

**Parameters**

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

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

**Returns**

half

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

__device__ __half __hdiv (const __half a, const __half b)
Performs half division in round-to-nearest-even mode.

Description
Divides half input a by input b in round-to-nearest mode.

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

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

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

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

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

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

Parameters
a
- half. Is only being read.

b
- half. Is only being read.

c
- half. Is only being read.

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

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

__device__ __half __hmul (const __half a, const __half b)
Performs half multiplication in round-to-nearest-even mode.

Description
Performs half multiplication of inputs a and b, in round-to-nearest mode.
__device__ __half __hmul_rn (const __half a, const __half b)
Performs half multiplication in round-to-nearest-even mode.

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

__device__ __half __hmul_sat (const __half a, const __half b)
Performs half multiplication in round-to-nearest-even mode, with saturation to [0.0, 1.0].

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

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

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

__device__ __half __hneg (const __half a)
Negates input half number and returns the result.

Description
Negates input half number and returns the result.

__device__ __half __hsub (const __half a, const __half b)
Performs half subtraction in round-to-nearest-even mode.

Description
Subtracts half input b from input a in round-to-nearest mode.
__device__ __half __hsub_rn (const __half a, const __half b)
Performs half subtraction in round-to-nearest-even mode.

Description
Subtracts half input b from input a in round-to-nearest mode. Prevents floating-point contractions of mul+sub into fma.

__device__ __half __hsub_sat (const __half a, const __half b)
Performs half subtraction in round-to-nearest-even mode, with saturation to [0.0, 1.0].

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

Returns
half
★ The result of subtraction of b from a, with respect to saturation.

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

__device__ __half atomicAdd (const __half *address, const __half val)
Adds val to the value stored at address in global or shared memory, and writes this value back to address. This operation is performed in one atomic operation.

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

- The old value read from address.

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

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

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.

```c
__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.

```c
__device__ __half2 __hadd2_rn (const __half2 a, const __half2 b)
```
Performs `half2` vector addition in round-to-nearest-even mode.

**Description**
Performs `half2` vector add of inputs a and b, in round-to-nearest mode. Prevents floating-point contractions of mul+add into fma.

```c
__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.
__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

__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

Perform vector fused multiply-add 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 \texttt{half2} vector multiply on inputs $a$ and $b$, then performs a \texttt{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__ \texttt{__half2 \_hmul2 (const \_half2 a, const \_half2 b)}

Performs \texttt{half2} vector multiplication in round-to-nearest-even mode.

**Description**

Performs \texttt{half2} vector multiplication of inputs $a$ and $b$, in round-to-nearest-even mode.

__device__ \texttt{__half2 \_hmul2\_rn (const \_half2 a, const \_half2 b)}

Performs \texttt{half2} vector multiplication in round-to-nearest-even mode.

**Description**

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

__device__ \texttt{__half2 \_hmul2\_sat (const \_half2 a, const \_half2 b)}

Performs \texttt{half2} vector multiplication in round-to-nearest-even mode, with saturation to $[0.0, 1.0]$.

**Parameters**

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

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

**Returns**

\texttt{half2}

- The result of elementwise multiplication of vectors $a$ and $b$, with respect to saturation.
Description
Performs half2 vector multiplication of inputs a and b, in round-to-nearest-even mode, and clamps the results to range [0.0, 1.0]. NaN results are flushed to +0.0.

__device__ __half2 __hneg2 (const __half2 a)
Negates both halves of the input half2 number and returns the result.

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

__device__ __half2 __hsub2 (const __half2 a, const __half2 b)
Performs half2 vector subtraction in round-to-nearest-even mode.

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

__device__ __half2 __hsub2_rn (const __half2 a, const __half2 b)
Performs half2 vector subtraction in round-to-nearest-even mode.

Description
Subtracts half2 input vector b from input vector a in round-to-nearest-even mode. Prevents floating-point contractions of mul+sub into fma.

__device__ __half2 __hsub2_sat (const __half2 a, const __half2 b)
Performs half2 vector subtraction in round-to-nearest-even mode, with saturation to [0.0, 1.0].

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

Returns
half2
The subtraction of vector \( b \) from \( a \), with respect to saturation.

**Description**

Subtracts 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.

```c
__device__ __half2 atomicAdd (const __half2 *address, const __half2 val)
```

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

**Parameters**

- **address**
  - __half2*. An address in global or shared memory.

- **val**
  - __half2. The value to be added.

**Returns**

- **half2**

  - The old value read from \( address \).

**Description**

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

**Note:**

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

### 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 half unordered greater-equal comparison.

Parameters

a
- half. Is only being read.

b
- half. Is only being read.

Returns

bool

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

Description

Performs half greater-equal comparison of inputs a and b. NaN inputs generate true results.
__device__ bool __hgt (const __half a, const __half b)
Performs half greater-than comparison.

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

Returns
bool

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

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

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

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

Returns
bool

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

Description
Performs half greater-than comparison of inputs a and b. NaN inputs generate true results.
__device__ int __hisinf (const __half a)
Checks if the input half number is infinite.

Parameters
a
- half. Is only being read.

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

Description
Checks if the input half number a is infinite.

__device__ bool __hisnan (const __half a)
Determine whether half argument is a NaN.

Parameters
a
- half. Is only being read.

Returns
bool
- true
  iff argument is NaN.

Description
Determine whether half value a is a NaN.
__device__ bool __hle (const __half a, const __half b)
Performs half less-equal comparison.

Parameters

a
- half. Is only being read.

b
- half. Is only being read.

Returns

bool

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

Description

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

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

Parameters

a
- half. Is only being read.

b
- half. Is only being read.

Returns

bool

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

Description

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

Parameters
a  
- half. Is only being read.

b  
- half. Is only being read.

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

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

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

Parameters
a  
- half. Is only being read.

b  
- half. Is only being read.

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

Description
Performs half less-than comparison of inputs a and b. NaN inputs generate true results.
__device__ __half __hmax (const __half a, const __half b)
Calculates half maximum of two input values.

Description
Calculates half max(a, b) defined as \( a \geq b \) ? 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 __hmax_nan (const __half a, const __half b)
Calculates half maximum of two input values, NaNs pass through.

Description
Calculates half max(a, b) defined as \( a \geq b \) ? 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__ __half __hmin (const __half a, const __half b)
Calculates half minimum of two input values.

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

```c
__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**
- `true` if both half results of if-equal comparison of vectors `a` and `b` are true;
- `false` otherwise.

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

```c
__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**
- `true` if both half results of unordered if-equal comparison of vectors `a` and `b` are true;
- `false` otherwise.
Performs \texttt{half2} vector if-equal comparison of inputs \(a\) and \(b\). The \texttt{bool} result is set to true only if both \texttt{half} if-equal comparisons evaluate to true, or false otherwise. NaN inputs generate true results.

\begin{verbatim}
\_device\_\_ bool \_ hbge2 (const \_ half2 a, const \_ half2 b)
\end{verbatim}

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

**Parameters**

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

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

**Returns**

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

**Description**

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

\begin{verbatim}
\_device\_\_ bool \_ hbgeu2 (const \_ half2 a, const \_ half2 b)
\end{verbatim}

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

**Parameters**

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

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

\texttt{__device__ bool __hbgt2 (const __half2 a, const __half2 b)}
Performs \texttt{half2} vector greater-than comparison and returns boolean true iff both \texttt{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 \(a\) and \(b\). The bool result is set to true only if both \texttt{half} greater-than comparisons evaluate to true, or false otherwise. NaN inputs generate false results.
__device__ bool __hbgtu2 (const __half2 a, const __half2 b)
Performs half2 vector unordered greater-than comparison and returns boolean true iff both half results are true, boolean false otherwise.

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

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

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

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

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

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

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

\[
\text{__device__ bool __hbleu2 (const __half2 a, const __half2 b)}
\]

Performs \( \text{half2} \) vector unordered less-equal comparison and returns boolean true iff both \( \text{half} \) results are true, boolean false otherwise.

**Parameters**

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

**Returns**

bool
- true
  - if both half results of unordered less-equal comparison of vectors \( a \) and \( b \) are true;
- false
  - otherwise.

**Description**

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

\[
\text{__device__ bool __hblt2 (const __half2 a, const __half2 b)}
\]

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

**Parameters**

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

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

**Description**

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

```c
__device__ bool __hbltu2 (const __half2 a, const __half2 b)
```

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

**Parameters**

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

**Returns**

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

**Description**

Performs half2 vector less-than comparison of inputs `a` and `b`. The bool result is set to true only if both half less-than comparisons evaluate to true, or false otherwise. NaN inputs generate true results.
__device__ bool __hbne2 (const __half2 a, const __half2 b)
Performs half2 vector not-equal comparison and returns boolean true iff both half results are true, boolean false otherwise.

Parameters

a
- half2. Is only being read.

b
- half2. Is only being read.

Returns

bool

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

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

__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

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

__device__ __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

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.

__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.

__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.

\[
\text{__device__ __half2 __hisnan2 (const __half2 a)}
\]
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.

\[
\text{__device__ __half2 __hle2 (const __half2 a, const __half2 b)}
\]
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.
__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

__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

__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 half2 vector unordered not-equal comparison.

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

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

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

1.1.5. Half Precision Conversion and Data Movement

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

__host____device__ __half __double2half (const double a)
Converts double number to half precision in round-to-nearest-even mode and returns half with converted value.

Parameters
a
- double. Is only being read.
Returns
half
  a
  converted to half.

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

__host____device__ __half2 __float22half2_rn (const float2 a)
Converts both components of float2 number to half precision in round-to-nearest-even mode and returns half2 with converted values.

Parameters
a
  - float2. Is only being read.

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

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

__host____device__ __half __float2half (const float a)
Converts float number to half precision in round-to-nearest-even mode and returns half with converted value.

Parameters
a
  - float. Is only being read.

Returns
half
a

converted to half.

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

__host____device__ __half2 __float2half2_rn (const float a)
Converts input to half precision in round-to-nearest-even mode and populates both halves of half2 with converted value.

Parameters
a
- float. Is only being read.

Returns
half2

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

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

__host____device__ __half __float2half_rd (const float a)
Converts float number to half precision in round-down mode and returns half with converted value.

Parameters
a
- float. Is only being read.

Returns
half

- a

converted to half.

Description
Converts float number a to half precision in round-down mode.
__host__ __device__ __half __float2half_rn (const float a)
Converts float number to half precision in round-to-nearest-even mode and returns half with converted value.

**Parameters**

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

**Returns**

- half

  - a converted to half.

**Description**

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

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

**Parameters**

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

**Returns**

- half

  - a converted to half.

**Description**

Converts float number `a` to half precision in round-up mode.
`__host__ __device__ __half __float2half_rz (const float a)`
Converts float number to half precision in round-towards-zero mode and returns `half` with converted value.

**Parameters**

`a`
- float. Is only being read.

**Returns**

`half`
- `a` converted to half.

**Description**

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

`__host__ __device__ __half2 __floats2half2_rn (const float a, const float b)`
Converts both input floats to half precision in round-to-nearest-even mode and returns `half2` with converted values.

**Parameters**

`a`
- float. Is only being read.

`b`
- float. Is only being read.

**Returns**

`half2`
- The `half2` value with corresponding halves equal to the converted input floats.

**Description**

Converts both input floats to half precision in round-to-nearest-even mode and combines the results into one `half2` number. Low 16 bits of the return value correspond to the input `a`, high 16 bits correspond to the input `b`. 
__host__ __device__ float2 __half22float2 (const __half2 a)
Converts both halves of half2 to float2 and returns the result.

Parameters
a
- half2. Is only being read.

Returns
float2
› a
converted to float2.

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

__host__ __device__ float __half2float (const __half a)
Converts half number to float.

Parameters
a
- float. Is only being read.

Returns
float
› a
converted to float.

Description
Converts half number a to float.

__device__ __half2 __half2half2 (const __half a)
Returns half2 with both halves equal to the input value.

Parameters
a
- half. Is only being read.
Returns

half2

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

Description

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

_device__ int __half2int_rd (const __half h)

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

Parameters

h

- half. Is only being read.

Returns

int

- h converted to a signed integer.

Description

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

_device__ int __half2int_rn (const __half h)

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

Parameters

h

- half. Is only being read.

Returns

int

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

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

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

Returns
int
\( h \)
 converted to a signed integer.

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

\[
\_\_host\_\_\_device\_\_ int \_\_half2int\_rz (const \_\_half h)
\]
Convert a half to a signed integer in round-towards-zero mode.

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

Returns
int
\( h \)
 converted to a signed integer.

Description
Convert the half-precision floating-point value \( h \) to a signed integer in round-towards-zero mode. NaN inputs are converted to 0.
__device__ long long int __half2ll_rd (const __half h)
Convert a half to a signed 64-bit integer in round-down mode.

Parameters
h
- half. Is only being read.

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

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

__device__ long long int __half2ll_rn (const __half h)
Convert a half to a signed 64-bit integer in round-to-nearest-even mode.

Parameters
h
- half. Is only being read.

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

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

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

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

- h
  converted to a signed 64-bit integer.

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

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

Parameters
h
- half. Is only being read.

Returns
long long int

- h
  converted to a signed 64-bit integer.

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

__device__ short int __half2short_rd (const __half h)
Convert a half to a signed short integer in round-down mode.

Parameters
h
- half. Is only being read.

Returns
short int

- h
converted to a signed short integer.

**Description**

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

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

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

**Parameters**

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

**Returns**

- short int

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

**Description**

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

```
__device__ short int __half2short_ru (const __half h)
```

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

**Parameters**

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

**Returns**

- short int

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

**Description**

Convert the half-precision floating-point value \( h \) to a signed short integer in round-up mode. NaN inputs are converted to 0.
__host____device__ short int __half2short_rz (const __half h)
Convert a half to a signed short integer in round-towards-zero mode.

Parameters
h
- half. Is only being read.

Returns
short int
  h
  converted to a signed short integer.

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

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

Parameters
h
- half. Is only being read.

Returns
unsigned int
  h
  converted to an unsigned integer.

Description
Convert the half-precision floating-point value \( h \) to an unsigned integer in round-down mode. NaN inputs are converted to 0.
`__device__ unsigned int __half2uint_rn (const __half h)`
Convert a half to an unsigned integer in round-to-nearest-even mode.

**Parameters**

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

**Returns**

- unsigned int
  - `h` converted to an unsigned integer.

**Description**

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

`__device__ unsigned int __half2uint_ru (const __half h)`
Convert a half to an unsigned integer in round-up mode.

**Parameters**

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

**Returns**

- unsigned int
  - `h` converted to an unsigned integer.

**Description**

Convert the half-precision floating-point value `h` to an unsigned integer in round-up mode. NaN inputs are converted to 0.
__host__ __device__ unsigned int __half2uint_rz (const __half h)
Convert a half to an unsigned integer in round-towards-zero mode.

Parameters
h
- half. Is only being read.

Returns
unsigned int
  h
  converted to an unsigned integer.

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

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

Parameters
h
- half. Is only being read.

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

Description
Convert the half-precision floating-point value h to an unsigned 64-bit integer in round-down mode. NaN inputs return 0x8000000000000000.
__device__ unsigned long long int __half2ull_rn (const __half h)
Convert a half to an unsigned 64-bit integer in round-to-nearest-even mode.

Parameters

h
- half. Is only being read.

Returns

unsigned long long int

h
converted to an unsigned 64-bit integer.

Description

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

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

Parameters

h
- half. Is only being read.

Returns

unsigned long long int

h
converted to an unsigned 64-bit integer.

Description

Convert the half-precision floating-point value h to an unsigned 64-bit integer in round-up mode. NaN inputs return 0x8000000000000000.
__host__ __device__ unsigned long long int __half2ull_rz (const __half h)
Convert a half to an unsigned 64-bit integer in round-towards-zero mode.

Parameters
h
  - half. Is only being read.

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

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

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

Parameters
h
  - half. Is only being read.

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

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

Parameters
h
  - half. Is only being read.

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

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

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

Parameters
h
  - half. Is only being read.

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

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

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

Parameters

h
- half. Is only being read.

Returns

unsigned short int

- h converted to an unsigned short integer.

Description

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

__device__ short int __half_as_short (const __half h)

Reinterprets bits in a half as a signed short integer.

Parameters

h
- half. Is only being read.

Returns

short int

- The reinterpreted value.

Description

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

Parameters
h
- half. Is only being read.

Returns
unsigned short int
  ▶ The
    reinterpreted value.

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

__device__ __half2 __halves2half2 (const __half a, const __half b)
Combines two half numbers into one half2 number.

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

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

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

Parameters
a
- half2. Is only being read.

Returns
float

- The high 16 bits of a converted to float.

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

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

Parameters
a
- half2. Is only being read.

Returns
half

- The high 16 bits of the input.

Description
Returns high 16 bits of half2 input a.

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

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

The half2 with both halves equal to the high 16 bits of the input.

Description
Extracts high 16 bits from half2 input a and returns a new half2 number which has both halves equal to the extracted bits.

__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 (const 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.

\[ \_\text{host}\_\text{device}\_\text{half} \_\text{int2half\_rn} \left( \text{const int } i \right) \]
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.

\[ \_\text{device}\_\text{half} \_\text{int2half\_ru} \left( \text{const int } i \right) \]
Convert a signed integer to a half in round-up mode.

Parameters
\[ i \]
- int. Is only being read.

Returns
half

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

\[
\text{__device__ __half __int2half_rz (const int } i
\]
Convert a signed integer to a half in round-towards-zero mode.

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

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

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

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

Parameters
\( ptr \)
- memory location

Returns
The value pointed by `ptr`

\[
\text{__device__ __half2 __ldca (const __half2 *ptr)}
\]
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 (const long long int i)
Convert a signed 64-bit integer to a half in round-down mode.

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

Returns
half
  i
  converted to half.

Description
Convert the signed 64-bit integer value i to a half-precision floating-point value in round-down mode.
__host__ __device__ __half __ll2half_rn (const long long int i)
Convert a signed 64-bit integer to a half in round-to-nearest-even mode.

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

Returns
half
  i
  converted to half.

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

__device__ __half __ll2half_ru (const long long int i)
Convert a signed 64-bit integer to a half in round-up mode.

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

Returns
half
  i
  converted to half.

Description
Convert the signed 64-bit integer value i to a half-precision floating-point value in round-up mode.
__device__ __half __ll2half_rz (const long long int i)
Convert a signed 64-bit integer to a half in round-towards-zero mode.

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

Returns
half
  - i converted to half.

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

__host____device__ float __low2float (const __half2 a)
Converts low 16 bits of half2 to float and returns the result.

Parameters
a
  - half2. Is only being read.

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

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

__device__ __half __low2half (const __half2 a)
Returns low 16 bits of half2 input.

Parameters
a
  - half2. Is only being read.
Returns

half

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

Description

Returns low 16 bits of half2 input a.

__device__ __half2 __low2half2 (const __half2 a)

Extracts low 16 bits from half2 input.

Parameters

a
  
  - half2. Is only being read.

Returns

half2

- The
  
  half2 with both halves equal to the low 16 bits of the input.

Description

Extracts low 16 bits from half2 input a and returns a new half2 number which has both halves equal to the extracted bits.

__device__ __half2 __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 half2 input and returns a new half2 number with swapped halves.

```c
__device__ __half2 __lows2half2 (const __half2 a, const __half2 b)
```

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

**Parameters**

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

**Returns**

half2

The low 16 bits of `a` and of `b`.

**Description**

Extracts low 16 bits from each of the two half2 inputs and combines into one half2 number. Low 16 bits from input `a` is stored in low 16 bits of the return value, low 16 bits from input `b` is stored in high 16 bits of the return value.

```c
__device__ __half __shfl_down_sync (const unsigned mask, const __half var, const unsigned int delta, const int width)
```

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

**Parameters**

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

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

**Description**

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

---

**Note:**

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

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

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

**Parameters**

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

**Returns**

Returns the 4-byte word referenced by var from the source thread ID as half2. If the source thread ID is out of range or the source thread has exited, the calling thread’s own var is returned.
Description
Calculates a source thread ID by adding delta to the caller's thread ID. The value of var held by the resulting thread ID is returned: this has the effect of shifting var down the warp by delta threads. If width is less than warpSize then each subsection of the warp behaves as a separate entity with a starting logical thread ID of 0. As for __shfl_up_sync(), the ID number of the source thread will not wrap around the value of width and so the upper delta threads will remain unchanged.

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

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

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

Description
Returns the value of var held by the thread whose ID is given by delta. If width is less than warpSize then each subsection of the warp behaves as a separate entity with a starting logical thread ID of 0. If delta is outside the range [0:width-1], the value returned corresponds to the value of var held by the delta modulo width (i.e. within the same subsection). width must have a value which is a power of 2; results are undefined if width is not a power of 2, or is a number greater than warpSize.
__device__ __half2 __shfl_sync (const unsigned mask, const __half2 var, const int delta, const int width)
Exchange a variable between threads within a warp. Direct copy from indexed thread.

Parameters

mask
- unsigned int. Is only being read.

var
- half2. Is only being read.

delta
- int. Is only being read.

width
- int. Is only being read.

Returns

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

Description

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

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

Parameters

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

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

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

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

__device__ __half2 __shfl_up_sync (const unsigned mask, const __half2 var, const unsigned int delta, const int width)
Exchange a variable between threads within a warp. Copy from a thread with lower ID relative to the caller.

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

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

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

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

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

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

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

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

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

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

Parameters

mask
- unsigned int. Is only being read.

var
- 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 (const short int i)
Convert a signed short integer to a half in round-down mode.

Parameters
i
  - short int. Is only being read.

Returns
half
  i
  converted to half.

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

__host__ __device__ __half __short2half_rn (const short int i)
Convert a signed short integer to a half in round-to-nearest-even mode.

Parameters
i
  - short int. Is only being read.

Returns
half
  i
  converted to half.

Description
Convert the signed short integer value i to a half-precision floating-point value in round-to-nearest-even mode.
__device__ __half __short2half_ru (const short int i)
Convert a signed short integer to a half in round-up mode.

Parameters
i
  - short int. Is only being read.

Returns
half
  i
  converted to half.

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

__device__ __half __short2half_rz (const short int i)
Convert a signed short integer to a half in round-towards-zero mode.

Parameters
i
  - short int. Is only being read.

Returns
half
  i
  converted to half.

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

__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.

\[ \texttt{__device__ void __stcg (const __half *ptr, const __half value)} \]
Generates a `st.global.cg` store instruction.

Parameters
- \( \text{ptr} \) - memory location
- \( \text{value} \) - the value to be stored

\[ \texttt{__device__ void __stcg (const __half2 *ptr, const __half2 value)} \]
Generates a `st.global.cg` store instruction.

Parameters
- \( \text{ptr} \) - memory location
- \( \text{value} \) - the value to be stored

\[ \texttt{__device__ void __stcs (const __half *ptr, const __half value)} \]
Generates a `st.global.cs` store instruction.

Parameters
- \( \text{ptr} \) - memory location
- \( \text{value} \) - the value to be stored
__device__ void __stcs (const __half2 *ptr, const __half2 value)
Generates a `st.global.cs` store instruction.

Parameters

ptr
- memory location

value
- the value to be stored

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

Parameters

ptr
- memory location

value
- the value to be stored

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

Parameters

ptr
- memory location

value
- the value to be stored

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

Parameters

ptr
- memory location

value
- the value to be stored

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

Parameters

ptr
- memory location

value
- the value to be stored

__device__ __half __uint2half_rd (const unsigned int i)
Convert an unsigned integer to a half in round-down mode.

Parameters

i
- unsigned int. Is only being read.

Returns

half

i
- converted to half.

Description

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

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

Parameters

i
- unsigned int. Is only being read.

Returns

half

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

\texttt{__device\_\_half \_uint2half\_ru (const unsigned int i)}
Convert an unsigned integer to a half in round-up mode.

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

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

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

\texttt{__device\_\_half \_uint2half\_rz (const unsigned int i)}
Convert an unsigned integer to a half in round-towards-zero mode.

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

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

Description
Convert the unsigned integer value \( i \) to a half-precision floating-point value in round-towards-zero mode.
__device__ __half __ull2half_rd (const unsigned long long int i)
Convert an unsigned 64-bit integer to a half in round-down mode.

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

Returns
half
  ▶ i
  converted to half.

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

__host__ __device__ __half __ull2half_rn (const unsigned long long int i)
Convert an unsigned 64-bit integer to a half in round-to-nearest-even mode.

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

Returns
half
  ▶ i
  converted to half.

Description
Convert the unsigned 64-bit integer value \( i \) to a half-precision floating-point value in round-to-nearest-even mode.
__device__ __half __ull2half_ru (const unsigned long long int i)
Convert an unsigned 64-bit integer to a half in round-up mode.

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

Returns
half
  i
  converted to half.

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

__device__ __half __ull2half_rz (const unsigned long long int i)
Convert an unsigned 64-bit integer to a half in round-towards-zero mode.

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

Returns
half
  i
  converted to half.

Description
Convert the unsigned 64-bit integer value \( i \) to a half-precision floating-point value in round-towards-zero mode.
__device__ __half __ushort2half_rd (const unsigned short int i)
Convert an unsigned short integer to a half in round-down mode.

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

Returns
half
  - i converted to half.

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

__host____device__ __half __ushort2half_rn (const unsigned short int i)
Convert an unsigned short integer to a half in round-to-nearest-even mode.

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

Returns
half
  - i converted to half.

Description
Convert the unsigned short integer value \( i \) to a half-precision floating-point value in round-to-nearest-even mode.
__device__ __half __ushort2half_ru (const unsigned short int i)
Convert an unsigned short integer to a half in round-up mode.

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

Returns
half
- i converted to half.

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

__device__ __half __ushort2half_rz (const unsigned short int i)
Convert an unsigned short integer to a half in round-towards-zero mode.

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

Returns
half
- i converted to half.

Description
Convert the unsigned short integer value i to a half-precision floating-point value in round-towards-zero mode.
__device__ __half __ushort_as_half (const unsigned short int i)

Reinterprets bits in an unsigned short integer as a half.

Parameters

i
- unsigned short int. Is only being read.

Returns

half
- The reinterpreted value.

Description

Reinterprets the bits in the unsigned short integer i as a half-precision floating-point number.

1.1.6. Half Math Functions

Half Precision Intrinsics

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

__device__ __half hceil (const __half h)

Calculate ceiling of the input argument.

Parameters

h
- half. Is only being read.

Returns

half
- The smallest integer value not less than h.

Description

Compute the smallest integer value not less than h.
__device__ __half hcos (const __half a)
Calculates half cosine in round-to-nearest-even mode.

Parameters

a
- half. Is only being read.

Returns
half

- The cosine of a.

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

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

Parameters

a
- half. Is only being read.

Returns
half

- The natural exponential function on a.

Description
Calculates half natural exponential function of input a in round-to-nearest-even mode.

__device__ __half hexp10 (const __half a)
Calculates half decimal exponential function in round-to-nearest mode.

Parameters

a
- half. Is only being read.
Returns

half

- The decimal exponential function on a.

Description

Calculates half decimal exponential function of input a in round-to-nearest-even mode.

__device__ __half hexp2 (const __half a)

Calculates half binary exponential function in round-to-nearest mode.

Parameters

a

- half. Is only being read.

Returns

half

- The binary exponential function on a.

Description

Calculates half binary exponential function of input a in round-to-nearest-even mode.

__device__ __half hfloor (const __half h)

Calculate the largest integer less than or equal to h.

Parameters

h

- half. Is only being read.

Returns

half

- The largest integer value which is less than or equal to h.
Description
Calculate the largest integer value which is less than or equal to h.

__device__ __half hlog (const __half a)
Calculates half natural logarithm in round-to-nearest-even mode.

Parameters
a
- half. Is only being read.

Returns
half
▶ The
   natural logarithm of a.

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

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

Parameters
a
- half. Is only being read.

Returns
half
▶ The
   decimal logarithm of a.

Description
Calculates half decimal logarithm of input a in round-to-nearest-even mode.
__device__ __half hlog2 (const __half a)
Calculates half binary logarithm in round-to-nearest-even mode.

Parameters
a
- half. Is only being read.

Returns
half
  - The binary logarithm of a.

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

__device__ __half hrcp (const __half a)
Calculates half reciprocal in round-to-nearest-even mode.

Parameters
a
- half. Is only being read.

Returns
half
  - The reciprocal of a.

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

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

Parameters
h
- half. Is only being read.
Returns
half
- The nearest integer to h.

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

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

Parameters
a
- half. Is only being read.

Returns
half
- The reciprocal square root of a.

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

__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.

```cuda
__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 \texttt{half2} binary exponential function of input vector \(a\) in round-to-nearest-even mode.

\texttt{__device__ __half2 h2floor (const __half2 h)}
Calculate the largest integer less than or equal to \(h\).

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

Returns
\texttt{half2}
\begin{itemize}
  \item The vector of largest integers which is less than or equal to \(h\).
\end{itemize}

Description
For each component of vector \(h\) calculate the largest integer value which is less than or equal to \(h\).

\texttt{__device__ __half2 h2log (const __half2 a)}
Calculates \texttt{half2} vector natural logarithm in round-to-nearest-even mode.

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

Returns
\texttt{half2}
\begin{itemize}
  \item The elementwise natural logarithm on vector \(a\).
\end{itemize}

Description
Calculates \texttt{half2} natural logarithm of input vector \(a\) in round-to-nearest-even mode.
__device__ __half2 h2log10 (const __half2 a)
Calculates half2 vector decimal logarithm in round-to-nearest-even mode.

Parameters
a
- half2. Is only being read.

Returns
half2
  - The elementwise decimal logarithm on vector a.

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

\texttt{\_\_device\_\_half2 h2rint (const \_\_half2 h)}
Round input to nearest integer value in half-precision floating-point number.

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

Returns
half2

- The vector of rounded integer values.

Description
Round each component of \texttt{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 \texttt{half2} vector reciprocal square root in round-to-nearest mode.

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

Returns
half2

- The elementwise reciprocal square root on vector a.
Description
Calculates half2 reciprocal square root of input vector a in round-to-nearest-even mode.

__device__ __half2 h2sin (const __half2 a)
Calculates half2 vector sine in round-to-nearest-even mode.

Parameters
a
- half2. Is only being read.

Returns
half2

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

__device__ __half2 h2sqrt (const __half2 a)
Calculates half2 vector square root in round-to-nearest-even mode.

Parameters
a
- half2. Is only being read.

Returns
half2

Description
Calculates half2 square root of input vector a in round-to-nearest mode.
__device__ __half2 h2trunc (const __half2 h)
Truncate half2 vector input argument to the integral part.

Parameters
h
- half2. Is only being read.

Returns
half2
- The truncated h.

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

1.2. 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.

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

Calculates the absolute value of input `nv_bfloat16` number and returns the result.

**Parameters**

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

**Returns**

- `nv_bfloat16`

  - The absolute value of `a`.

**Description**

Calculates the absolute value of input `nv_bfloat16` number and returns the result.

```c
__device__ __nv_bfloat16 __hadd (const __nv_bfloat16 a, const __nv_bfloat16 b)
```

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

**Description**

Performs `nv_bfloat16` addition of inputs `a` and `b`, in round-to-nearest-even mode.
__device__ __nv_bfloat16 __hadd_rn (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs \textit{nv\_bfloat16} addition in round-to-nearest-even mode.

\textbf{Description}

Performs \textit{nv\_bfloat16} addition of inputs \textit{a} and \textit{b}, in round-to-nearest-even mode. Prevents floating-point contractions of mul+add into fma.

__device__ __nv_bfloat16 __hadd_sat (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs \textit{nv\_bfloat16} addition in round-to-nearest-even mode, with saturation to \([0.0, 1.0]\).

\textbf{Parameters}

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

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

\textbf{Returns}

\textit{nv\_bfloat16}

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

\textbf{Description}

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

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

\textbf{Description}

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

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

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

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

Returns
\texttt{nv\_bfloat16}
- The result of fused multiply-add operation on \(a\), \(b\), and \(c\) with relu saturation.

Description
Performs \texttt{nv\_bfloat16} multiply on inputs \(a\) and \(b\), then performs a \texttt{nv\_bfloat16} add of the result with \(c\), rounding the result once in round-to-nearest-even mode. 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 \( \text{nv\_bfloa}_{16} \) fused multiply-add in round-to-nearest-even mode, with saturation to \([0.0, 1.0]\).

**Parameters**

**a**

- \( \text{nv\_bfloa}_{16} \). Is only being read.

**b**

- \( \text{nv\_bfloa}_{16} \). Is only being read.

**c**

- \( \text{nv\_bfloa}_{16} \). Is only being read.

**Returns**

\( \text{nv\_bfloa}_{16} \)

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

**Description**

Performs \( \text{nv\_bfloa}_{16} \) multiply on inputs \( a \) and \( b \), then performs a \( \text{nv\_bfloa}_{16} \) add of the result with \( c \), rounding the result once in round-to-nearest-even mode, and clamps the result to range \([0.0, 1.0]\). NaN results are flushed to +0.0.

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

Performs \( \text{nv\_bfloa}_{16} \) multiplication in round-to-nearest-even mode.

**Description**

Performs \( \text{nv\_bfloa}_{16} \) multiplication of inputs \( a \) and \( b \), in round-to-nearest mode.

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

Performs \( \text{nv\_bfloa}_{16} \) multiplication in round-to-nearest-even mode.

**Description**

Performs \( \text{nv\_bfloa}_{16} \) multiplication of inputs \( a \) and \( b \), in round-to-nearest mode. Prevents floating-point contractions of mul+add or sub into fma.
__device__ __nv_bfloat16 __hmul_sat (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs \texttt{nv\_bfloat16} multiplication in round-to-nearest-even mode, with saturation to [0.0, 1.0].

**Parameters**

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

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

**Returns**
\texttt{nv\_bfloat16}

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

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

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

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

__device__ __nv_bfloat16 __hsub (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs \texttt{nv\_bfloat16} subtraction in round-to-nearest-even mode.

**Description**
Subtracts \texttt{nv\_bfloat16} input \texttt{b} from input \texttt{a} in round-to-nearest mode.
__device__ __nv_bfloat16 __hsub_rn (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs \texttt{nv\_bfloat16} subtraction in round-to-nearest-even mode.

**Description**
Subtracts \texttt{nv\_bfloat16} input \(b\) from input \(a\) in round-to-nearest mode. Prevents floating-point contractions of mul+sub into fma.

__device__ __nv_bfloat16 __hsub_sat (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs \texttt{nv\_bfloat16} subtraction in round-to-nearest-even mode, with saturation to [0.0, 1.0].

**Parameters**

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

**Returns**
\texttt{nv\_bfloat16}

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

**Description**
Subtracts \texttt{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.

__device__ __nv_bfloat16 atomicAdd (const __nv_bfloat16 *address, const __nv_bfloat16 val)
Adds \(val\) to the value stored at \texttt{address} in global or shared memory, and writes this value back to \texttt{address}. This operation is performed in one atomic operation.

**Parameters**

- \texttt{address}\ - \texttt{__nv\_bfloat16*}. An address in global or shared memory.
- \texttt{val}\ - \texttt{__nv\_bfloat16}. The value to be added.
Returns
__nv_bfloat16

- The old value read from address.

Description
The location of address must be in global or shared memory. This operation has undefined behavior otherwise. This operation is only supported by devices of compute capability 8.x and higher.

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

1.2.2. 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 nv_bfloat162 number and returns the result.
__device__ __nv_bfloat162 __hadd2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs nv_bfloat162 vector addition in round-to-nearest-even mode.

Description
Performs nv_bfloat162 vector add of inputs a and b, in round-to-nearest mode.

__device__ __nv_bfloat162 __hadd2_rn (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs nv_bfloat162 vector addition in round-to-nearest-even mode.

Description
Performs nv_bfloat162 vector add of inputs a and b, in round-to-nearest mode. Prevents floating-point contractions of mul+add into fma.

__device__ __nv_bfloat162 __hadd2_sat (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs nv_bfloat162 vector addition in round-to-nearest-even mode, with saturation to [0.0, 1.0].

Parameters
a
- nv_bfloat162. Is only being read.

b
- nv_bfloat162. Is only being read.

Returns
nv_bfloat162
- The sum of a and b, with respect to saturation.

Description
Performs nv_bfloat162 vector add of inputs a and b, in round-to-nearest mode, and clamps the results to range [0.0, 1.0]. NaN results are flushed to +0.0.
__device__ __nv_bfloat162 __hcmadd (const __nv_bfloat162 a, const __nv_bfloat162 b, const __nv_bfloat162 c)
Performs fast complex multiply-accumulate.

Parameters

a
- __nv_bfloat162. Is only being read.

b
- __nv_bfloat162. Is only being read.

c
- __nv_bfloat162. Is only being read.

Returns

__nv_bfloat162

The result of complex multiply-accumulate operation on complex numbers a, b, and c.

Description

Interprets vector __nv_bfloat162 input pairs a, b, and c as complex numbers in __nv_bfloat16 precision and performs complex multiply-accumulate operation: a*b + c.

__device__ __nv_bfloat162 __hfma2 (const __nv_bfloat162 a, const __nv_bfloat162 b, const __nv_bfloat162 c)
Performs __nv_bfloat162 vector fused multiply-add in round-to-nearest-even mode.

Description

Performs __nv_bfloat162 vector multiply on inputs a and b, then performs a __nv_bfloat162 vector add of the result with c, rounding the result once in round-to-nearest-even mode.

__device__ __nv_bfloat162 __hfma2_relu (const __nv_bfloat162 a, const __nv_bfloat162 b, const __nv_bfloat162 c)
Performs __nv_bfloat162 vector fused multiply-add in round-to-nearest-even mode with relu saturation.

Parameters

a
- __nv_bfloat162. Is only being read.
b
- nv_bfloat162. Is only being read.

c
- nv_bfloat162. Is only being read.

Returns
nv_bfloat162

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

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

__device__ __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 \texttt{nv\_bfloat162} vector multiply on inputs \(a\) and \(b\), then performs a \texttt{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.

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

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

\begin{verbatim}
__device__ __nv_bfloat162 __hmul2_rn (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs \texttt{nv\_bfloat162} vector multiplication in round-to-nearest-even mode. Prevents floating-point contractions of mul+add or sub into fma.
\end{verbatim}

\begin{verbatim}
__device__ __nv_bfloat162 __hmul2_sat (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs \texttt{nv\_bfloat162} vector multiplication in round-to-nearest-even mode, with saturation to \([0.0, 1.0]\).
\end{verbatim}

Parameters
\begin{itemize}
  \item \(a\)
    \begin{itemize}
      \item \texttt{nv\_bfloat162}. Is only being read.
    \end{itemize}
  \item \(b\)
    \begin{itemize}
      \item \texttt{nv\_bfloat162}. Is only being read.
    \end{itemize}
\end{itemize}

Returns
\texttt{nv\_bfloat162}

- The result of elementwise multiplication of vectors \(a\) and \(b\), with respect to saturation.
Description
Performs `nv_bfloat162` vector multiplication of inputs `a` and `b`, in round-to-nearest-even mode, and clamps the results to range \([0.0, 1.0]\). NaN results are flushed to +0.0.

```c
__device__ __nv_bfloat162 __hneg2 (const __nv_bfloat162 a)
```
Negates both halves of the input `nv_bfloat162` number and returns the result.

Description
Negates both halves of the input `nv_bfloat162` number `a` and returns the result.

```c
__device__ __nv_bfloat162 __hsub2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
```
Performs `nv_bfloat162` vector subtraction in round-to-nearest-even mode.

Description
Subtracts `nv_bfloat162` input vector `b` from input vector `a` in round-to-nearest-even mode.

```c
__device__ __nv_bfloat162 __hsub2_rn (const __nv_bfloat162 a, const __nv_bfloat162 b)
```
Performs `nv_bfloat162` vector subtraction in round-to-nearest-even mode.

Description
Subtracts `nv_bfloat162` input vector `b` from input vector `a` in round-to-nearest-even mode. Prevents floating-point contractions of mul+sub into fma.

```c
__device__ __nv_bfloat162 __hsub2_sat (const __nv_bfloat162 a, const __nv_bfloat162 b)
```
Performs `nv_bfloat162` vector subtraction in round-to-nearest-even mode, with saturation to \([0.0, 1.0]\).

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

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

Description
Subtracts \( \text{nv\_bfloat162} \) 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.

\[
\text{\_\_device\_\_nv\_bfloat162 atomicAdd (const \_\_nv\_bfloat162 *address, const \_\_nv\_bfloat162 val)}
\]
Vector add \( val \) to the value stored at \( address \) in global or shared memory, and writes this value back to \( address \). The atomicity of the add operation is guaranteed separately for each of the two \( \text{nv\_bfloat16} \) elements; the entire \( \_\_nv\_bfloat162 \) is not guaranteed to be atomic as a single 32-bit access.

Parameters

\text{address}
- \( \_\_nv\_bfloat162* \). An address in global or shared memory.

\text{val}
- \( \_\_nv\_bfloat162 \). The value to be added.

Returns
\( \_\_nv\_bfloat162 \)

- The old value read from \( address \).

Description
The location of \( address \) must be in global or shared memory. This operation has undefined behavior otherwise. This operation is only supported by devices of compute capability 8.x and higher.

Note:
For more details for this function see the Atomic Functions section in the CUDA C++ Programming Guide.
1.2.3. Bfloat16 Comparison Functions

Bfloat16 Precision Intrinsics

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

```c
__device__ bool __heq (const __nv_bfloat16 a, const __nv_bfloat16 b)
```
Performs `nv_bfloat16` if-equal comparison.

**Parameters**

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

**Returns**

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

**Description**

Performs `nv_bfloat16` if-equal comparison of inputs `a` and `b`. NaN inputs generate false results.

```c
__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 \texttt{nv\_bfloa16} if-equal comparison of inputs \texttt{a} and \texttt{b}. NaN inputs generate true results.

\begin{verbatim}
__device__ bool __hge (const __nv\_bfloa16 a, const __nv\_bfloa16 b)
\end{verbatim}
Performs \texttt{nv\_bfloa16} greater-equal comparison.

Parameters
\begin{itemize}
  \item \texttt{a} - \texttt{nv\_bfloa16}. Is only being read.
  \item \texttt{b} - \texttt{nv\_bfloa16}. Is only being read.
\end{itemize}

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

Description
Performs \texttt{nv\_bfloa16} greater-equal comparison of inputs \texttt{a} and \texttt{b}. NaN inputs generate false results.

\begin{verbatim}
__device__ bool __hgeu (const __nv\_bfloa16 a, const __nv\_bfloa16 b)
\end{verbatim}
Performs \texttt{nv\_bfloa16} unordered greater-equal comparison.

Parameters
\begin{itemize}
  \item \texttt{a} - \texttt{nv\_bfloa16}. Is only being read.
  \item \texttt{b} - \texttt{nv\_bfloa16}. Is only being read.
\end{itemize}

Returns
\begin{verbatim}
bool
\end{verbatim}
\begin{itemize}
  \item The boolean result of unordered greater-equal comparison of \texttt{a} and \texttt{b}.
\end{itemize}
Description
Performs `nv_bfloat16` greater-equal comparison of inputs `a` and `b`. NaN inputs generate true results.

```c
__device__ bool __hgt (const __nv_bfloat16 a, const __nv_bfloat16 b)
```
Performs `nv_bfloat16` greater-than comparison.

**Parameters**

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

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

**Returns**

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

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

```c
__device__ bool __hgtu (const __nv_bfloat16 a, const __nv_bfloat16 b)
```
Performs `nv_bfloat16` unordered greater-than comparison.

**Parameters**

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

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

**Returns**

`bool`
- The boolean result of unordered greater-than comparison of `a` and `b`. 
Description
Performs `nv_bfloat16` greater-than comparison of inputs `a` and `b`. NaN inputs generate true results.

```c
__device__ int __hisinf (const __nv_bfloat16 a)
```
Checks if the input `nv_bfloat16` number is infinite.

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

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

Description
Checks if the input `nv_bfloat16` number `a` is infinite.

```c
__device__ bool __hisnan (const __nv_bfloat16 a)
```
Determine whether `nv_bfloat16` argument is a NaN.

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

Returns
`bool`
- `true` iff argument is NaN.
Description
Determine whether \texttt{nv\_bfloat16} value \texttt{a} is a NaN.

\texttt{\_device\_ bool \_hle (const \_nv\_bfloat16 a, const \_nv\_bfloat16 b)}
Performs \texttt{nv\_bfloat16} less-equal comparison.

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

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

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

\texttt{\_device\_ bool \_hleu (const \_nv\_bfloat16 a, const \_nv\_bfloat16 b)}
Performs \texttt{nv\_bfloat16} unordered less-equal comparison.

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

Returns
\texttt{bool}
- The boolean result of unordered less-equal comparison of \texttt{a} and \texttt{b}.
**Description**
Performs `nv_bfloat16` less-equal comparison of inputs `a` and `b`. NaN inputs generate true results.

```c
__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.

```c
__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`. 

CUDA Math API
Description
Performs \texttt{nv\_bfloat16} less-than comparison of inputs \texttt{a} and \texttt{b}. NaN inputs generate true results.

\texttt{__device__ __nv\_bfloat16 __hmax (const __nv\_bfloat16 a, const __nv\_bfloat16 b)}
Calculates \texttt{nv\_bfloat16} maximum of two input values.

Description
Calculates \texttt{nv\_bfloat16} max[\texttt{a, b}] defined as (\texttt{a > b}) ? \texttt{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

\texttt{__device__ __nv\_bfloat16 __hmax\_nan (const __nv\_bfloat16 a, const __nv\_bfloat16 b)}
Calculates \texttt{nv\_bfloat16} maximum of two input values, NaNs pass through.

Description
Calculates \texttt{nv\_bfloat16} max[\texttt{a, b}] defined as (\texttt{a > b}) ? \texttt{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

\texttt{__device__ __nv\_bfloat16 __hmin (const __nv\_bfloat16 a, const __nv\_bfloat16 b)}
Calculates \texttt{nv\_bfloat16} minimum of two input values.

Description
Calculates \texttt{nv\_bfloat16} min[\texttt{a, b}] defined as (\texttt{a < b}) ? \texttt{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 \( \text{nv\_bfloat16} \) minimum of two input values, NaNs pass through.

**Description**
Calculates \( \text{nv\_bfloat16} \) min\(\{a, b\}\) defined as \((a < b) \ ? \ a : b\).

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

__device__ bool __hne (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs \( \text{nv\_bfloat16} \) not-equal comparison.

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

**Returns**
- \( \text{bool} \)
  - The boolean result of not-equal comparison of \( a \) and \( b \).

**Description**
Performs \( \text{nv\_bfloat16} \) not-equal comparison of inputs \( a \) and \( b \). NaN inputs generate false results.

__device__ bool __hneu (const __nv_bfloat16 a, const __nv_bfloat16 b)
Performs \( \text{nv\_bfloat16} \) unordered not-equal comparison.

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

bool

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

Description

Performs \(nv\_bfloat16\) not-equal comparison of inputs \(a\) and \(b\). NaN inputs generate true results.

1.2.4. Bfloat162 Comparison Functions

Bfloat16 Precision Intrinsics

To use these functions, include the header file \(cuda\_bf16.h\) in your program.

\[
\text{\_device\_bool \_hbeq2 (const \_nv\_bfloat162 a, const \_nv\_bfloat162 b)}
\]

Performs \(nv\_bfloat162\) vector if-equal comparison and returns boolean true iff both \(nv\_bfloat16\) results are true, boolean false otherwise.

Parameters

\(a\)

- \(nv\_bfloat162\). Is only being read.

\(b\)

- \(nv\_bfloat162\). Is only being read.

Returns

bool

- true
  
  if both \(nv\_bfloat16\) results of if-equal comparison of vectors \(a\) and \(b\) are true;

- false
  
  otherwise.

Description

Performs \(nv\_bfloat162\) vector if-equal comparison of inputs \(a\) and \(b\). The bool result is set to true only if both \(nv\_bfloat16\) if-equal comparisons evaluate to true, or false otherwise. NaN inputs generate false results.
__device__ bool __hbequ2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs \texttt{nv\_bfloat162} vector unordered if-equal comparison and returns boolean true iff both \texttt{nv\_bfloat16} results are true, boolean false otherwise.

Parameters
\begin{itemize}
  \item \texttt{a} \hspace{1cm} \texttt{- nv\_bfloat162. Is only being read.}
  \item \texttt{b} \hspace{1cm} \texttt{- nv\_bfloat162. Is only being read.}
\end{itemize}

Returns
bool
\begin{itemize}
  \item \texttt{true} \hspace{1cm} \texttt{if both \texttt{nv\_bfloat16} results of unordered if-equal comparison of vectors \texttt{a} and \texttt{b} are true;}
  \item \texttt{false} \hspace{1cm} \texttt{otherwise.}
\end{itemize}

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

__device__ bool __hbge2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs \texttt{nv\_bfloat162} vector greater-equal comparison and returns boolean true iff both \texttt{nv\_bfloat16} results are true, boolean false otherwise.

Parameters
\begin{itemize}
  \item \texttt{a} \hspace{1cm} \texttt{- nv\_bfloat162. Is only being read.}
  \item \texttt{b} \hspace{1cm} \texttt{- nv\_bfloat162. Is only being read.}
\end{itemize}

Returns
bool
\begin{itemize}
  \item \texttt{true}
\end{itemize}
if both \texttt{nv\_bf\textfloat基调16} results of greater-equal comparison of vectors \texttt{a} and \texttt{b} are true;

\begin{itemize}
  \item false
  \item otherwise.
\end{itemize}

\textbf{Description}

Performs \texttt{nv\_bf\textfloat基调16} vector greater-equal comparison of inputs \texttt{a} and \texttt{b}. The bool result is set to true only if both \texttt{nv\_bf\textfloat基调16} greater-equal comparisons evaluate to true, or false otherwise. NaN inputs generate false results.

\begin{verbatim}
__device__ bool __hbgeu2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
\end{verbatim}

Performs \texttt{nv\_bf\textfloat基调16} vector unordered greater-equal comparison and returns boolean true iff both \texttt{nv\_bf\textfloat基调16} results are true, boolean false otherwise.

\textbf{Parameters}

\begin{itemize}
  \item \texttt{a}
    - \texttt{nv\_bf\textfloat基调16}. Is only being read.
  \item \texttt{b}
    - \texttt{nv\_bf\textfloat基调16}. Is only being read.
\end{itemize}

\textbf{Returns}

bool

\begin{itemize}
  \item true
    if both \texttt{nv\_bf\textfloat基调16} results of unordered greater-equal comparison of vectors \texttt{a} and \texttt{b} are true;
  \item false
    otherwise.
\end{itemize}

\textbf{Description}

Performs \texttt{nv\_bf\textfloat基调16} vector greater-equal comparison of inputs \texttt{a} and \texttt{b}. The bool result is set to true only if both \texttt{nv\_bf\textfloat基调16} 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;
- false
  otherwise.

Description

Performs __nv_bfloat162 vector greater-than comparison of inputs a and b. The bool result is set to true only if both __nv_bfloat16 greater-than comparisons evaluate to true, or false otherwise. NaN inputs generate false results.

__device__ bool __hbgtu2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs __nv_bfloat162 vector unordered greater-than comparison and returns boolean true iff both __nv_bfloat16 results are true, boolean false otherwise.

Parameters

a
- __nv_bfloat162. Is only being read.

b
- __nv_bfloat162. Is only being read.

Returns

bool

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

- \texttt{false}
  
- otherwise.

**Description**

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

\texttt{__device\_ bool __hble2 (const __nv\_bfloat16\_2 a, const __nv\_bfloat16\_2 b)}

Performs \texttt{nv\_bfloat16} vector less-equal comparison and returns boolean true iff both \texttt{nv\_bfloat16} results are true, boolean false otherwise.

**Parameters**

- \texttt{a}
  - \texttt{nv\_bfloat16\_2}. Is only being read.

- \texttt{b}
  - \texttt{nv\_bfloat16\_2}. Is only being read.

**Returns**

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

**Description**

Performs \texttt{nv\_bfloat16\_2} vector less-equal comparison of inputs \texttt{a} and \texttt{b}. The bool result is set to true only if both \texttt{nv\_bfloat16} less-equal comparisons evaluate to true, or false otherwise. NaN inputs generate false results.
__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
- true
  - if both __nv_bfloat16 results of unordered less-equal comparison of vectors a and b are true;
- false
  - otherwise.

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 __nv_bfloat162 vector less-than comparison and returns boolean true iff both __nv_bfloat16 results are true, boolean false otherwise.

Parameters
a
- __nv_bfloat162. Is only being read.
b
- __nv_bfloat162. Is only being read.

Returns
bool
- true
if both \( \text{nv\_bf\_float16} \) results of less-than comparison of vectors \( a \) and \( b \) are true;

- false
  otherwise.

**Description**

Performs \( \text{nv\_bf\_float16} \) vector less-than comparison of inputs \( a \) and \( b \). The bool result is set to true only if both \( \text{nv\_bf\_float16} \) less-than comparisons evaluate to true, or false otherwise. NaN inputs generate false results.

\[ \text{__device\_bool __hbltu2 (const __nv\_bf\_float16 \textit{a}, const __nv\_bf\_float16 \textit{b})} \]

Performs \( \text{nv\_bf\_float16} \) vector unordered less-than comparison and returns boolean true iff both \( \text{nv\_bf\_float16} \) results are true, boolean false otherwise.

**Parameters**

- \( a \)
  - \( \text{nv\_bf\_float16} \). Is only being read.
- \( b \)
  - \( \text{nv\_bf\_float16} \). Is only being read.

**Returns**

bool

- true
  if both \( \text{nv\_bf\_float16} \) results of unordered less-than comparison of vectors \( a \) and \( b \) are true;
- false
  otherwise.

**Description**

Performs \( \text{nv\_bf\_float16} \) vector less-than comparison of inputs \( a \) and \( b \). The bool result is set to true only if both \( \text{nv\_bf\_float16} \) less-than comparisons evaluate to true, or false otherwise. NaN inputs generate true results.
__device__ bool __hbne2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs `nv_bfloat162` vector not-equal comparison and returns boolean `true` iff both `nv_bfloat16` results are true, boolean `false` otherwise.

Parameters

a
- `nv_bfloat162`. Is only being read.

b
- `nv_bfloat162`. Is only being read.

Returns

bool

- `true`
  - if both `nv_bfloat16` results of not-equal comparison of vectors `a` and `b` are true,
- `false`
  - otherwise.

Description

Performs `nv_bfloat162` vector not-equal comparison of inputs `a` and `b`. The bool result is set to `true` only if both `nv_bfloat16` not-equal comparisons evaluate to `true`, or `false` otherwise. NaN inputs generate false results.

__device__ bool __hbneu2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs `nv_bfloat162` vector unordered not-equal comparison and returns boolean `true` iff both `nv_bfloat16` results are true, boolean `false` otherwise.

Parameters

a
- `nv_bfloat162`. Is only being read.

b
- `nv_bfloat162`. Is only being read.

Returns

bool

- `true`
if both `nv_bfloat16` results of unordered not-equal comparison of vectors `a` and `b` are true;

- false
  otherwise.

**Description**

Performs `nv_bfloat16` 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_bfloat16` 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_bfloat16` 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.

```c
__device__ __nv_bfloat162 __hequ2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
```

Performs `nv_bfloat16` vector unordered if-equal comparison.

**Parameters**

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

\[ \text{The vector result of unordered if-equal comparison of vectors } \mathbf{a} \text{ and } \mathbf{b}. \]

Description

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

\[ \_ \_ \text{ device } \_ \_ \text{ nv\_bf\_float162 } \_ \_ \text{ hge2 } (\text{const } \_ \_ \text{ nv\_bf\_float162 } \_ \_ \text{ a}, \text{const } \_ \_ \text{ nv\_bf\_float162 } \_ \_ \text{ b}) \]

Performs \( \text{nv\_bf\_float162} \) vector greater-equal comparison.

Parameters

\( \mathbf{a} \)

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

\( \mathbf{b} \)

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

Returns

\[ \text{nv\_bf\_float162} \]

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

Description

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

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

```c
__device__ __nv_bfloat162 __hgtu2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
```
Performs `nv_bfloat162` vector unordered greater-than comparison.

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

Returns
`nv_bfloat162`

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

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

```c
__device__ __nv_bfloat162 __hisnan2 (const __nv_bfloat162 a)
```
Determine whether `nv_bfloat162` argument is a NaN.

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

Returns
`nv_bfloat162`

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

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

**Description**

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

```c
__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.

```c
__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 \( \text{nv}_\text{bf} \text{loat16}\text{2} \) vector less-equal comparison of inputs \( a \) and \( b \). The corresponding \( \text{nv}_\text{bf} \text{loat16} \) results are set to 1.0 for true, or 0.0 for false. NaN inputs generate true results.

\[
\_\text{device}\_\_\_\text{nv}_\text{bf} \text{loat16}\text{2} \_\text{hlt2} \left(\text{const} \_\_\_\text{nv}_\text{bf} \text{loat16}\text{2} \ a, \\
\text{const} \_\_\_\text{nv}_\text{bf} \text{loat16}\text{2} \ b\right)
\]

Performs \( \text{nv}_\text{bf} \text{loat16}\text{2} \) vector less-than comparison.

**Parameters**

\( a \)
- \( \text{nv}_\text{bf} \text{loat16}\text{2} \). Is only being read.

\( b \)
- \( \text{nv}_\text{bf} \text{loat16}\text{2} \). Is only being read.

**Returns**

\( \text{nv}_\text{bf} \text{loat16}\text{2} \)

The \( \text{nv}_\text{bf} \text{loat16}\text{2} \) vector result of less-than comparison of vectors \( a \) and \( b \).

**Description**

Performs \( \text{nv}_\text{bf} \text{loat16}\text{2} \) vector less-than comparison of inputs \( a \) and \( b \). The corresponding \( \text{nv}_\text{bf} \text{loat16} \) results are set to 1.0 for true, or 0.0 for false. NaN inputs generate false results.

\[
\_\text{device}\_\_\_\text{nv}_\text{bf} \text{loat16}\text{2} \_\text{hlru2} \left(\text{const} \_\_\_\text{nv}_\text{bf} \text{loat16}\text{2} \ a, \\
\text{const} \_\_\_\text{nv}_\text{bf} \text{loat16}\text{2} \ b\right)
\]

Performs \( \text{nv}_\text{bf} \text{loat16}\text{2} \) vector unordered less-than comparison.

**Parameters**

\( a \)
- \( \text{nv}_\text{bf} \text{loat16}\text{2} \). Is only being read.

\( b \)
- \( \text{nv}_\text{bf} \text{loat16}\text{2} \). Is only being read.
Returns

nv_bfloat162

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

Description

Performs \( \text{nv.bfloat162} \) vector less-than comparison of inputs \( a \) and \( b \). The corresponding \( \text{nv.bfloat16} \) results are set to 1.0 for true, or 0.0 for false. NaN inputs generate true results.

\[
\text{__device__ __nv_bfloat162 __hmax2 (const __nv_bfloat162 a, const __nv_bfloat162 b)}
\]

Calculates \( \text{nv.bfloat162} \) vector maximum of two inputs.

Description

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

\[
\text{__device__ __nv_bfloat162 __hmax2_nan (const __nv_bfloat162 a, const __nv_bfloat162 b)}
\]

Calculates \( \text{nv.bfloat162} \) vector maximum of two inputs, NaNs pass through.

Description

Calculates \( \text{nv.bfloat162} \) vector \( \text{max}(a, b) \). Elementwise \( \text{nv.bfloat16} \) 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 $\text{nv\_bf\_float162}$ vector minimum of two inputs.

**Description**
Calculates $\text{nv\_bf\_float162}$ vector min($a$, $b$). Elementwise $\text{nv\_bf\_float16}$ 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 $\text{nv\_bf\_float162}$ vector minimum of two inputs, NaNs pass through.

**Description**
Calculates $\text{nv\_bf\_float162}$ vector min($a$, $b$). Elementwise $\text{nv\_bf\_float16}$ 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 $\text{nv\_bf\_float162}$ vector not-equal comparison.

**Parameters**
- $a$ - $\text{nv\_bf\_float162}$. Is only being read.
- $b$ - $\text{nv\_bf\_float162}$. Is only being read.

**Returns**
- $\text{nv\_bf\_float162}$
- The vector result of not-equal comparison of vectors $a$ and $b$. 
Description
Performs \texttt{nv\_bf\texttt{f}loat162} vector not-equal comparison of inputs \texttt{a} and \texttt{b}. The corresponding \texttt{nv\_bf\texttt{f}loat16} results are set to 1.0 for true, or 0.0 for false. NaN inputs generate false results.

\begin{verbatim}
__device__ __nv_bfloat162 __hneu2 (const __nv_bfloat162 a, const __nv_bfloat162 b)
Performs \texttt{nv\_bf\texttt{f}loat162} vector unordered not-equal comparison.
\end{verbatim}

Parameters
\texttt{a} - \texttt{nv\_bf\texttt{f}loat162}. Is only being read.
\texttt{b} - \texttt{nv\_bf\texttt{f}loat162}. Is only being read.

Returns
\texttt{nv\_bf\texttt{f}loat162}

\begin{verbatim}
^ The
\end{verbatim}

vector result of unordered not-equal comparison of vectors \texttt{a} and \texttt{b}.

1.2.5. Bfloat16 Precision Conversion and Data Movement

Bfloat16 Precision Intrinsics
To use these functions, include the header file \texttt{cuda\_bf16.h} in your program.

\begin{verbatim}
__host____device__ float2 __bf\texttt{f}loat1622float2 (const __nv_bfloat162 a)
Converts both halves of \texttt{nv\_bf\texttt{f}loat162} to float2 and returns the result.
\end{verbatim}

Parameters
\texttt{a} - \texttt{nv\_bf\texttt{f}loat162}. Is only being read.
Returns
float2

- \( a \)
  converted to float2.

Description
Converts both halves of \( \text{nv\_bfloat16} \) input \( a \) to float2 and returns the result.

\[
\_\_\text{device\_\_nv\_bfloat16} \_\_\text{bfloat16}2\_\text{bfloat16} (\text{const} \_\_\text{nv\_bfloat16} a)
\]
Returns \( \text{nv\_bfloat16}2 \) with both halves equal to the input value.

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

Returns
\( \text{nv\_bfloat16}2 \)

- The
  vector which has both its halves equal to the input \( a \).

Description
Returns \( \text{nv\_bfloat16}2 \) number with both halves equal to the input \( \text{nv\_bfloat16} \) number.

\[
\_\_\text{host\_\_device\_\_float} \_\_\text{bfloat16}2\_\text{float} (\text{const} \_\_\text{nv\_bfloat16} a)
\]
Converts \( \text{nv\_bfloat16} \) number to float.

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

Returns
float

- \( a \)
converted to float.

**Description**

Converts `nv_bfloat16` number `a` to float.

```c
__device__ int __bfloat162int_rd (const __nv_bfloat16 h)
```

Convert a `nv_bfloat16` to a signed integer in round-down mode.

**Parameters**

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

**Returns**

`int`

- `h` converted to a signed integer.

**Description**

Convert the `nv_bfloat16` floating-point value `h` to a signed integer in round-down mode. NaN inputs are converted to 0.

```c
__device__ int __bfloat162int_rn (const __nv_bfloat16 h)
```

Convert a `nv_bfloat16` to a signed integer in round-to-nearest-even mode.

**Parameters**

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

**Returns**

`int`

- `h` converted to a signed integer.

**Description**

Convert the `nv_bfloat16` floating-point value `h` to a signed integer in round-to-nearest-even mode. NaN inputs are converted to 0.
__device__ int __bfloat162int_ru (const __nv_bfloat16 h)
Convert a __nv_bfloat16 to a signed integer in round-up mode.

Parameters

h
- __nv_bfloat16. Is only being read.

Returns

int

  h
  converted to a signed integer.

Description

Convert the __nv_bfloat16 floating-point value h to a signed integer in round-up mode. NaN inputs are converted to 0.

__host____device__ int __bfloat162int_rz (const __nv_bfloat16 h)
Convert a __nv_bfloat16 to a signed integer in round-towards-zero mode.

Parameters

h
- __nv_bfloat16. Is only being read.

Returns

int

  h
  converted to a signed integer.

Description

Convert the __nv_bfloat16 floating-point value h to a signed integer in round-towards-zero mode. NaN inputs are converted to 0.
__device__ long long int __bfloat162ll_rd (const __nv_bfloat16 h)
Convert a nv_bfloat16 to a signed 64-bit integer in round-down mode.

Parameters
h
- nv_bfloat16. Is only being read.

Returns
long long int

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

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

Parameters
h
- nv_bfloat16. Is only being read.

Returns
long long int

Description
Convert the nv_bfloat16 floating-point value \( h \) to a signed 64-bit integer in round-to-nearest-even mode. NaN inputs return a long long int with hex value of 0x8000000000000000.
**__device__ long long int __bfloat162ll_ru (const __nv_bfloat16 h)**

Convert a `nv_bfloat16` to a signed 64-bit integer in round-up mode.

**Parameters**

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

**Returns**

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

**Description**

Convert the `nv_bfloat16` floating-point value `h` to a signed 64-bit integer in round-up mode. NaN inputs return a long long int with hex value of 0x8000000000000000.

**__host__device__ long long int __bfloat162ll_rz (const __nv_bfloat16 h)**

Convert a `nv_bfloat16` to a signed 64-bit integer in round-towards-zero mode.

**Parameters**

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

**Returns**

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

**Description**

Convert the `nv_bfloat16` floating-point value `h` to a signed 64-bit integer in round-towards-zero mode. NaN inputs return a long long int with hex value of 0x8000000000000000.
\texttt{\_device\_short int \_bfloat162short\_rd (const \_nv\_bfloa16 h)}

Convert a \texttt{nv\_bfloa16} to a signed short integer in round-down mode.

\textbf{Parameters}

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

\textbf{Returns}

\texttt{short int}

\- \texttt{h} converted to a signed short integer.

\textbf{Description}

Convert the \texttt{nv\_bfloa16} floating-point value \texttt{h} to a signed short integer in round-down mode. NaN inputs are converted to 0.

\texttt{\_device\_short int \_bfloat162short\_rn (const \_nv\_bfloa16 h)}

Convert a \texttt{nv\_bfloa16} to a signed short integer in round-to-nearest-even mode.

\textbf{Parameters}

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

\textbf{Returns}

\texttt{short int}

\- \texttt{h} converted to a signed short integer.

\textbf{Description}

Convert the \texttt{nv\_bfloa16} floating-point value \texttt{h} to a signed short integer in round-to-nearest-even mode. NaN inputs are converted to 0.
__device__ short int __bfloat162short_ru (const __nv_bfloat16 h)
Convert a nv_bfloat16 to a signed short integer in round-up mode.

Parameters
h
- nv_bfloat16. Is only being read.

Returns
short int
  h
  converted to a signed short integer.

Description
Convert the nv_bfloat16 floating-point value h to a signed short integer in round-up mode. NaN inputs are converted to 0.

__host____device__ short int __bfloat162short_rz (const __nv_bfloat16 h)
Convert a nv_bfloat16 to a signed short integer in round-towards-zero mode.

Parameters
h
- nv_bfloat16. Is only being read.

Returns
short int
  h
  converted to a signed short integer.

Description
Convert the nv_bfloat16 floating-point value h to a signed short integer in round-towards-zero mode. NaN inputs are converted to 0.
__device__ unsigned int __bfloat162uint_rd (const __nv_bfloat16 h)
Convert a nv_bfloat16 to an unsigned integer in round-down mode.

Parameters
h
- nv_bfloat16. Is only being read.

Returns
unsigned int

h converted to an unsigned integer.

Description
Convert the nv_bfloat16 floating-point value h to an unsigned integer in round-down mode. NaN inputs are converted to 0.

__device__ unsigned int __bfloat162uint_rn (const __nv_bfloat16 h)
Convert a nv_bfloat16 to an unsigned integer in round-to-nearest-even mode.

Parameters
h
- nv_bfloat16. Is only being read.

Returns
unsigned int

h converted to an unsigned integer.

Description
Convert the nv_bfloat16 floating-point value h to an unsigned integer in round-to-nearest-even mode. NaN inputs are converted to 0.
__device__ unsigned int __bfloat162uint_ru (const __nv_bfloat16 h)
Convert a nv_bfloat16 to an unsigned integer in round-up mode.

Parameters
h
  - nv_bfloat16. Is only being read.

Returns
unsigned int
  ▶ h
  converted to an unsigned integer.

Description
Convert the nv_bfloat16 floating-point value h to an unsigned integer in round-up mode. NaN inputs are converted to 0.

__host____device__ unsigned int __bfloat162uint_rz (const __nv_bfloat16 h)
Convert a nv_bfloat16 to an unsigned integer in round-towards-zero mode.

Parameters
h
  - nv_bfloat16. Is only being read.

Returns
unsigned int
  ▶ h
  converted to an unsigned integer.

Description
Convert the nv_bfloat16 floating-point value h to an unsigned integer in round-towards-zero mode. NaN inputs are converted to 0.
__device__ unsigned long long int __bfloat162ull_rd (const __nv_bfloat16 h)
Convert a nv_bfloat16 to an unsigned 64-bit integer in round-down mode.

Parameters
h
- nv_bfloat16. Is only being read.

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

Description
Convert the nv_bfloat16 floating-point value h to an unsigned 64-bit integer in round-down mode. NaN inputs return 0x8000000000000000.

__device__ unsigned long long int __bfloat162ull_rn (const __nv_bfloat16 h)
Convert a nv_bfloat16 to an unsigned 64-bit integer in round-to-nearest-even mode.

Parameters
h
- nv_bfloat16. Is only being read.

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

Description
Convert the nv_bfloat16 floating-point value h to an unsigned 64-bit integer in round-to-nearest-even mode. NaN inputs return 0x8000000000000000.
__device__ unsigned long long int __bfloat162ull_ru (const __nv_bfloat16 h)
Convert a nv_bfloat16 to an unsigned 64-bit integer in round-up mode.

Parameters
h - nv_bfloat16. Is only being read.

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

Description
Convert the nv_bfloat16 floating-point value h to an unsigned 64-bit integer in round-up mode. NaN inputs return 0x8000000000000000.

__host__ __device__ unsigned long long int __bfloat162ull_rz (const __nv_bfloat16 h)
Convert a nv_bfloat16 to an unsigned 64-bit integer in round-towards-zero mode.

Parameters
h - nv_bfloat16. Is only being read.

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

Description
Convert the nv_bfloat16 floating-point value h to an unsigned 64-bit integer in round-towards-zero mode. NaN inputs return 0x8000000000000000.
__device__ unsigned short int __bfloat162ushort_rd (const __nv_bfloat16 h)
Convert a nv_bfloat16 to an unsigned short integer in round-down mode.

Parameters

h
- nv_bfloat16. Is only being read.

Returns

unsigned short int

h converted to an unsigned short integer.

Description

Convert the nv_bfloat16 floating-point value h to an unsigned short integer in round-down mode. NaN inputs are converted to 0.

__device__ unsigned short int __bfloat162ushort_rn (const __nv_bfloat16 h)
Convert a nv_bfloat16 to an unsigned short integer in round-to-nearest-even mode.

Parameters

h
- nv_bfloat16. Is only being read.

Returns

unsigned short int

h converted to an unsigned short integer.

Description

Convert the nv_bfloat16 floating-point value h to an unsigned short integer in round-to-nearest-even mode. NaN inputs are converted to 0.
__device__ unsigned short int __bfloat162ushort_ru (const __nv_bfloat16 h)
Convert a nv_bfloat16 to an unsigned short integer in round-up mode.

Parameters

h
- nv_bfloat16. Is only being read.

Returns

unsigned short int

h
converted to an unsigned short integer.

Description

Convert the nv_bfloat16 floating-point value h to an unsigned short integer in round-up mode. NaN inputs are converted to 0.

__host___device__ unsigned short int __bfloat162ushort_rz (const __nv_bfloat16 h)
Convert a nv_bfloat16 to an unsigned short integer in round-towards-zero mode.

Parameters

h
- nv_bfloat16. Is only being read.

Returns

unsigned short int

h
converted to an unsigned short integer.

Description

Convert the nv_bfloat16 floating-point value h to an unsigned short integer in round-towards-zero mode. NaN inputs are converted to 0.
**__device__ short int __bfloat16_as_short (const __nv_bfloat16 h)**

Reinterprets bits in a `nv_bfloat16` as a signed short integer.

**Parameters**

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

**Returns**

- short int
  - The reinterpreted value.

**Description**

Reinterprets the bits in the `nv_bfloat16` floating-point number `h` as a signed short integer.

**__device__ unsigned short int __bfloat16_as_ushort (const __nv_bfloat16 h)**

Reinterprets bits in a `nv_bfloat16` as an unsigned short integer.

**Parameters**

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

**Returns**

- unsigned short int
  - The reinterpreted value.

**Description**

Reinterprets the bits in the `nv_bfloat16` floating-point number `h` as an unsigned short number.
__host____device__ __nv_bfloat16 __double2bfloat16 (const double a)
Converts double number to nv_bfloat16 precision in round-to-nearest-even mode and returns nv_bfloat16 with converted value.

Parameters
a
- double. Is only being read.

Returns
nv_bfloat16
▶ a
converted to nv_bfloat16.

Description
Converts double number a to nv_bfloat16 precision in round-to-nearest-even mode.

__host____device__ __nv_bfloat162 __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 __float2bf16 (const float a)
Converts float number to __nv_bfloat16 precision in round-to-nearest-even mode and returns
__nv_bfloat16 with converted value.

Parameters

a
- float. Is only being read.

Returns

__nv_bfloat16

- a
  converted to __nv_bfloat16.

Description

Converts float number a to __nv_bfloat16 precision in round-to-nearest-even mode.

__host____device__ __nv_bfloat162 __float2bf162_rn (const float a)
Converts input to __nv_bfloat16 precision in round-to-nearest-even mode and populates both
halves of __nv_bfloat162 with converted value.

Parameters

a
- float. Is only being read.

Returns

__nv_bfloat162

- The
  __nv_bfloat162 value with both halves equal to the converted __nv_bfloat16 precision
  number.

Description

Converts input a to __nv_bfloat16 precision in round-to-nearest-even mode and populates both
halves of __nv_bfloat162 with converted value.
__host__ __device__ __nv_bfloat16 __float2bfloat16_rd
(const float a)
Converts float number to nv_bfloat16 precision in round-down mode and returns
nv_bfloat16 with converted value.

Parameters
a
  - float. Is only being read.

Returns
nv_bfloat16
  a
  converted to nv_bfloat16.

Description
Converts float number a to nv_bfloat16 precision in round-down mode.

__host__ __device__ __nv_bfloat16 __float2bfloat16_rn
(const float a)
Converts float number to nv_bfloat16 precision in round-to-nearest-even mode and returns
nv_bfloat16 with converted value.

Parameters
a
  - float. Is only being read.

Returns
nv_bfloat16
  a
  converted to nv_bfloat16.

Description
Converts float number a to nv_bfloat16 precision in round-to-nearest-even mode.
__host__ __device__ __nv_bfloat16 __float2bfloat16_ru (const float a)
Converting a 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**

Converting a float number `a` to `nv_bfloat16` precision in round-up mode.

__host__ __device__ __nv_bfloat16 __float2bfloat16_rz (const float a)
Converting a float number to `nv_bfloat16` precision in round-towards-zero mode and returns `nv_bfloat16` with converted value.

**Parameters**

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

**Returns**

- `nv_bfloat16`
  - a converted to `nv_bfloat16`.

**Description**

Converting a float number `a` to `nv_bfloat16` precision in round-towards-zero mode.
__host__ __device__ __nv_bfloat162 __floats2bfloat162_rn (const float a, const float b)

Converts both input floats to nv_bfloat16 precision in round-to-nearest-even mode and returns nv_bfloat162 with converted values.

Parameters

a
- float. Is only being read.

b
- float. Is only being read.

Returns

nv_bfloat162

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

Description

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

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

Combines two nv_bfloat16 numbers into one nv_bfloat162 number.

Parameters

a
- nv_bfloat16. Is only being read.

b
- nv_bfloat16. Is only being read.

Returns

nv_bfloat162

‣ The nv_bfloat162 with one nv_bfloat16 equal to a and the other to b.
Description
Combines two input \texttt{nv\_bfloa16} number \texttt{a} and \texttt{b} into one \texttt{nv\_bfloa16} number. Input \texttt{a} is stored in low 16 bits of the return value, input \texttt{b} is stored in high 16 bits of the return value.

\texttt{\_device\_\_nv\_bfloa16 \_high2bfloa16 (const \_nv\_bfloa162 a)}
Returns high 16 bits of \texttt{nv\_bfloa162} input.

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

Returns
\texttt{nv\_bfloa16}
- The high 16 bits of the input.

Description
Returns high 16 bits of \texttt{nv\_bfloa162} input \texttt{a}.

\texttt{\_device\_\_nv\_bfloa162 \_high2bfloa162 (const \_nv\_bfloa162 a)}
Extracts high 16 bits from \texttt{nv\_bfloa162} input.

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

Returns
\texttt{nv\_bfloa162}
- The \texttt{nv\_bfloa162} with both halves equal to the high 16 bits of the input.

Description
Extracts high 16 bits from \texttt{nv\_bfloa162} input \texttt{a} and returns a new \texttt{nv\_bfloa162} number which has both halves equal to the extracted bits.
__host__ __device__ float __high2float (const __nv_bfloat162 a)
Converts high 16 bits of nv_bfloat162 to float and returns the result.

Parameters

- **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_bfloat162 __highs2bfloat162 (const __nv_bfloat162 a, const __nv_bfloat162 b)
Extracts high 16 bits from each of the two nv_bfloat162 inputs and combines into one nv_bfloat162 number.

Parameters

- **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 (const int i)
Convert a signed integer to a nv_bfloat16 in round-down mode.

Parameters
i
- int. Is only being read.

Returns
nv_bfloat16

- i
  converted to nv_bfloat16.

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

__host____device__ __nv_bfloat16 __int2bfloat16_rn (const int i)
Convert a signed integer to a nv_bfloat16 in round-to-nearest-even mode.

Parameters
i
- int. Is only being read.

Returns
nv_bfloat16

- i
  converted to nv_bfloat16.

Description
Convert the signed integer value \( i \) to a nv_bfloat16 floating-point value in round-to-nearest-even mode.
__device__ __nv_bfloat16 __int2bfloat16_ru (const int i)
Convert a signed integer to a nv_bfloat16 in round-up mode.

Parameters
i
- int. Is only being read.

Returns
nv_bfloat16
- i converted to nv_bfloat16.

Description
Convert the signed integer value i to a nv_bfloat16 floating-point value in round-up mode.

__device__ __nv_bfloat16 __int2bfloat16_rz (const int i)
Convert a signed integer to a nv_bfloat16 in round-towards-zero mode.

Parameters
i
- int. Is only being read.

Returns
nv_bfloat16
- i converted to nv_bfloat16.

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

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

Parameters
ptr
- memory location
__device__ __nv_bfloat16 __ldca (const __nv_bfloat16 *ptr)
Generates a `ld.global.ca` load instruction.

Parameters

ptr
- memory location

Returns
The value pointed by `ptr`

__device__ __nv_bfloat16 __ldcg (const __nv_bfloat162 *ptr)
Generates a `ld.global.cg` load instruction.

Parameters

ptr
- memory location

Returns
The value pointed by `ptr`

__device__ __nv_bfloat162 __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_bfloat162 __ldcs (const __nv_bfloat162 *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

Returns
The value pointed by `ptr`

__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_bfloat162 __ldg (const __nv_bfloat162 *ptr)`
Generates a `ld.global.nc` load instruction.

Parameters
`ptr`
- memory location

Returns
The value pointed by `ptr`

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

Parameters
`ptr`
- memory location

Returns
The value pointed by `ptr`

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

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

```c
__device__ __nv_bfloat16 __ll2bfloat16_rd (const long long int i)
```
Convert a signed 64-bit integer to a `nv_bfloat16` in round-down mode.

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

Returns
`nv_bfloat16`
  - i converted to `nv_bfloat16`.

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

```c
__host__ __device__ __nv_bfloat16 __ll2bfloat16_rn (const long long int i)
```
Convert a signed 64-bit integer to a `nv_bfloat16` in round-to-nearest-even mode.

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

Returns
`nv_bfloat16`
  - i converted to `nv_bfloat16`.

Description
Convert the signed 64-bit integer value i to a `nv_bfloat16` floating-point value in round-to-nearest-even mode.
\texttt{\_\_device\_\_ \_nv\_bfloat16\ _\_ll2bfloat16\_ru (const long long int i)}

Convert a signed 64-bit integer to a \texttt{nv\_bfloat16} in round-up mode.

\textbf{Parameters}

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

\textbf{Returns}

\texttt{nv\_bfloat16}

\begin{itemize}
  \item \texttt{i} converted to \texttt{nv\_bfloat16}.
\end{itemize}

\textbf{Description}

Convert the signed 64-bit integer value \texttt{i} to a \texttt{nv\_bfloat16} floating-point value in round-up mode.

\texttt{\_\_device\_\_ \_nv\_bfloat16\ _\_ll2bfloat16\_rz (const long long int i)}

Convert a signed 64-bit integer to a \texttt{nv\_bfloat16} in round-towards-zero mode.

\textbf{Parameters}

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

\textbf{Returns}

\texttt{nv\_bfloat16}

\begin{itemize}
  \item \texttt{i} converted to \texttt{nv\_bfloat16}.
\end{itemize}

\textbf{Description}

Convert the signed 64-bit integer value \texttt{i} to a \texttt{nv\_bfloat16} floating-point value in round-towards-zero mode.
__device__ __nv_bfloat16 __low2bfloat16 (const __nv_bfloat162 a)

Returns low 16 bits of nv_bfloat162 input.

Parameters
a
- nv_bfloat162. Is only being read.

Returns
nv_bfloat16

Description
Returns low 16 bits of nv_bfloat162 input a.

__device__ __nv_bfloat162 __low2bfloat162 (const __nv_bfloat162 a)

Extracts low 16 bits from nv_bfloat162 input.

Parameters
a
- nv_bfloat162. Is only being read.

Returns
nv_bfloat162

Description
Extracts low 16 bits from nv_bfloat162 input a and returns a new nv_bfloat162 number which has both halves equal to the extracted bits.
__host__ __device__ float __low2float (const __nv_bfloat162 a)
Converts low 16 bits of nv_bfloat162 to float and returns the result.

Parameters

a
- nv_bfloat162. Is only being read.

Returns

float

- The low 16 bits of a converted to float.

Description

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

__device__ __nv_bfloat162 __lowhigh2highlow (const __nv_bfloat162 a)
Swaps both halves of the nv_bfloat162 input.

Parameters

a
- nv_bfloat162. Is only being read.

Returns

nv_bfloat162

- a with its halves being swapped.

Description

Swaps both halves of the nv_bfloat162 input and returns a new nv_bfloat162 number with swapped halves.
__device__ __nv_bfloat16 __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 (const unsigned mask, const __nv_bfloat16 var, const unsigned int delta, const int width)

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

Parameters

mask
- unsigned int. Is only being read.

var
- nv_bfloat16. Is only being read.

delta
- int. Is only being read.

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

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

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

__device__ __nv_bfloat162 __shfl_down_sync (const unsigned mask, const __nv_bfloat162 var, const unsigned int delta, const int width)
Exchange a variable between threads within a warp. Copy from a thread with higher ID relative to the caller.

Parameters

- **mask**
  - unsigned int. Is only being read.

- **var**
  - nv_bfloat162. Is only being read.

- **delta**
  - int. Is only being read.

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

Returns
Returns the 4-byte word referenced by var from the source thread ID as nv_bfloat162. If the source thread ID is out of range or the source thread has exited, the calling thread’s own var is returned.
Description
Calculates a source thread ID by adding delta to the caller’s thread ID. The value of var held by the resulting thread ID is returned: this has the effect of shifting var down the warp by delta threads. If width is less than warpSize then each subsection of the warp behaves as a separate entity with a starting logical thread ID of 0. As for \_shfl\_up\_sync[], the ID number of the source thread will not wrap around the value of width and so the upper delta threads will remain unchanged.

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

\_
device\_
\_\_nv\_bfloat16 \_shfl\_sync (const unsigned
mask, const \_\_nv\_bfloat16 var, const int delta, const int
width)
Exchange a variable between threads within a warp. Direct copy from indexed thread.

Parameters
- \_mask\_ - unsigned int. Is only being read.
- \_var\_ - nv\_bfloat16. Is only being read.
- \_delta\_ - int. Is only being read.
- \_width\_ - int. Is only being read.

Returns
Returns the 2-byte word referenced by var from the source thread ID as nv\_bfloat16. If the source thread ID is out of range or the source thread has exited, the calling thread’s own var is returned.

Description
Returns the value of var held by the thread whose ID is given by delta. If width is less than warpSize then each subsection of the warp behaves as a separate entity with a starting logical thread ID of 0. If delta is outside the range [0:width\_1], the value returned corresponds to the value of var held by the delta modulo width (i.e. within the same subsection). width must have
a value which is a power of 2; results are undefined if width is not a power of 2, or is a number greater than warpSize.

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

__device__ __nv_bfloat162 __shfl_sync (const unsigned mask, const __nv_bfloat162 var, const int delta, const int width)
Exchange a variable between threads within a warp. Direct copy from indexed thread.

Parameters

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

Returns

Returns the 4-byte word referenced by var from the source thread ID as __nv_bfloat162. If the source thread ID is out of range or the source thread has exited, the calling thread’s own var is returned.

Description

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

Note:
For more details for this function see the Warp Shuffle Functions section in the CUDA C++ Programming Guide.
__device__ __nv_bfloat16 __shfl_up_sync (const unsigned mask, const __nv_bfloat16 var, const unsigned int delta, const int width)
Exchange a variable between threads within a warp. Copy from a thread with lower ID relative to the caller.

Parameters

- **mask** - unsigned int. Is only being read.
- **var** - nv_bfloat16. Is only being read.
- **delta** - int. Is only being read.
- **width** - int. Is only being read.

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

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

Note:
For more details for this function see the Warp Shuffle Functions section in the CUDA C++ Programming Guide.
__device__ __nv_bfloat162 __shfl_up_sync (const unsigned mask, const __nv_bfloat162 var, const unsigned int delta, const int width)

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

Parameters

- **mask**
  - unsigned int. Is only being read.
- **var**
  - nv_bfloat162. Is only being read.
- **delta**
  - int. Is only being read.
- **width**
  - int. Is only being read.

Returns

Returns the 4-byte word referenced by var from the source thread ID as nv_bfloat162. If the source thread ID is out of range or the source thread has exited, the calling thread’s own var is returned.

Description

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

Note:

For more details for this function see the Warp Shuffle Functions section in the CUDA C++ Programming Guide.
__device__ __nv_bfloat16 __shfl_xor_sync (const unsigned mask, const __nv_bfloat16 var, const int delta, const int width)

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

Parameters

mask
- unsigned int. Is only being read.

var
- nv_bfloat16. Is only being read.

delta
- int. Is only being read.

width
- int. Is only being read.

Returns

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

Description

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

Note:

For more details for this function see the Warp Shuffle Functions section in the CUDA C++ Programming Guide.
__device__ __nv_bfloat16 __shfl_xor_sync (const unsigned mask, const __nv_bfloat16 var, const int delta, const int width)

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

Parameters

- **mask**
  - unsigned int. Is only being read.

- **var**
  - __nv_bfloat16. Is only being read.

- **delta**
  - int. Is only being read.

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

Returns

Returns the 4-byte word referenced by var from the source thread ID as __nv_bfloat16. If the source thread ID is out of range or the source thread has exited, the calling thread’s own var is returned.

Description

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

**Note:**

For more details for this function see the Warp Shuffle Functions section in the CUDA C++ Programming Guide.
__device__ __nv_bfloat16 __short2bfloat16_rd (const short int i)
Convert a signed short integer to a nv_bfloat16 in round-down mode.

Parameters
i
- short int. Is only being read.

Returns
nv_bfloat16
i
converted to nv_bfloat16.

Description
Convert the signed short integer value \( i \) to a \text{nv\_bf\_float16} \) floating-point value in round-down mode.

__host____device__ __nv_bfloat16 __short2bfloat16_rn (const short int i)
Convert a signed short integer to a \text{nv\_bf\_float16} \) 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 \text{nv\_bf\_float16} \) floating-point value in round-to-nearest-even mode.
__device__ __nv_bfloat16 __short2bfloat16_ru (const short int i)
Convert a signed short integer to a nv_bfloat16 in round-up mode.

Parameters
i
- short int. Is only being read.

Returns
nv_bfloat16
‣ i converted to nv_bfloat16.

Description
Convert the signed short integer value i to a nv_bfloat16 floating-point value in round-up mode.

__device__ __nv_bfloat16 __short2bfloat16_rz (const short int i)
Convert a signed short integer to a nv_bfloat16 in round-towards-zero mode.

Parameters
i
- short int. Is only being read.

Returns
nv_bfloat16
‣ i converted to nv_bfloat16.

Description
Convert the signed short integer value i to a nv_bfloat16 floating-point value in round-towards-zero mode.
__device__ __nv_bfloat16 __short_as_bfloat16 (const short int i)
Reinterprets bits in a signed short integer as a nv_bfloat16.

Parameters
i
- short int. Is only being read.

Returns
nv_bfloat16
- The reinterpreted value.

Description
Reinterprets the bits in the signed short integer `i` as a nv_bfloat16 floating-point number.

__device__ void __stcg (const __nv_bfloat16 *ptr, const __nv_bfloat16 value)
Generates a `st.global.cg` store instruction.

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

__device__ void __stcg (const __nv_bfloat162 *ptr, const __nv_bfloat162 value)
Generates a `st.global.cg` store instruction.

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

Parameters

ptr
- memory location

value
- the value to be stored

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

Parameters

ptr
- memory location

value
- the value to be stored

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

Parameters

ptr
- memory location

value
- the value to be stored

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

Parameters

ptr
- memory location

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

Parameters

ptr
- memory location
value
- the value to be stored

__device__ void __stwt (const __nv_bfloat162* ptr, const __nv_bfloat162 value)
Generates a `st.global.wt` store instruction.

Parameters

ptr
- memory location
value
- the value to be stored

__device__ __nv_bfloat16 __uint2bf16_rd (const unsigned int i)
Convert an unsigned integer to a nv_bfloat16 in round-down mode.

Parameters

i
- unsigned int. Is only being read.

Returns

nv_bfloat16

Description
Convert the unsigned integer value \( i \) to a \( \text{nv.bfloat16} \) floating-point value in round-down mode.
__host__ __device__ __nv_bfloat16 __uint2bfload16_rn
(const unsigned int i)
Convert an unsigned integer to a nv_bfloat16 in round-to-nearest-even mode.

Parameters
i
- unsigned int. Is only being read.

Returns
nv_bfloat16

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

__device__ __nv_bfloat16 __uint2bfload16_ru (const unsigned int i)
Convert an unsigned integer to a nv_bfloat16 in round-up mode.

Parameters
i
- unsigned int. Is only being read.

Returns
nv_bfloat16

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

Parameters

i
- unsigned int. Is only being read.

Returns

nv_bfloat16

i
converted to nv_bfloat16.

Description

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

__device__ __nv_bfloat16 __ull2bfloat16_rd (const unsigned long long int i)
Convert an unsigned 64-bit integer to a nv_bfloat16 in round-down mode.

Parameters

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

Returns

nv_bfloat16

i
converted to nv_bfloat16.

Description

Convert the unsigned 64-bit integer value \( i \) to a nv_bfloat16 floating-point value in round-down mode.
__host__ __device__ __nv_bfloat16 __ull2bfloat16_rn (const unsigned long long int i)
Convert an unsigned 64-bit integer to a nv_bfloat16 in round-to-nearest-even mode.

Parameters

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

Returns

nv_bfloat16

i
  converted to nv_bfloat16.

Description

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

__device__ __nv_bfloat16 __ull2bfloat16_ru (const unsigned long long int i)
Convert an unsigned 64-bit integer to a nv_bfloat16 in round-up mode.

Parameters

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

Returns

nv_bfloat16

i
  converted to nv_bfloat16.

Description

Convert the unsigned 64-bit integer value i to a nv_bfloat16 floating-point value in round-up mode.
__device__ __nv_bfloat16 __ull2bfloat16_rz (const unsigned long long int i)
Convert an unsigned 64-bit integer to a nv_bfloat16 in round-towards-zero mode.

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

Returns
nv_bfloat16
  i
    converted to nv_bfloat16.

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

__device__ __nv_bfloat16 __ushort2bfloat16_rd (const unsigned short int i)
Convert an unsigned short integer to a nv_bfloat16 in round-down mode.

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

Returns
nv_bfloat16
  i
    converted to nv_bfloat16.

Description
Convert the unsigned short integer value i to a nv_bfloat16 floating-point value in round-down mode.
__host__ __device__ __nv_bfloat16 __ushort2bfloat16_rn(const unsigned short int i)
Convert an unsigned short integer to a nv_bfloat16 in round-to-nearest-even mode.

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

Returns
nv_bfloat16

i
converted to nv_bfloat16.

Description
Convert the unsigned short integer value i to a nv_bfloat16 floating-point value in round-to-nearest-even mode.

__device__ __nv_bfloat16 __ushort2bfloat16_ru(const unsigned short int i)
Convert an unsigned short integer to a nv_bfloat16 in round-up mode.

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

Returns
nv_bfloat16

i
converted to nv_bfloat16.

Description
Convert the unsigned short integer value i to a nv_bfloat16 floating-point value in round-up mode.
__device__ __nv_bfloat16 __ushort2bf16_rz (const
unsigned short int i)
Convert an unsigned short integer to a nv_bfloat16 in round-towards-zero mode.

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

Returns
nv_bfloat16

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

__device__ __nv_bfloat16 __ushort_as_bfloat16 (const
unsigned short int i)
Reinterprets bits in an unsigned short integer as a nv_bfloat16.

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

Returns
nv_bfloat16

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

Description
Compute the smallest integer value not less than h.

__device__ __nv_bfloat16 hcos (const __nv_bfloat16 a)
Calculates nv_bfloat16 cosine in round-to-nearest-even mode.

Parameters
a
- nv_bfloat16. Is only being read.

Returns
nv_bfloat16

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

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

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

- The natural exponential function on \( a \).

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

\[
\text{__device__ __nv\_bfloat16 hexp10 (const __nv\_bfloat16 a)}
\]
Calculates \( \text{nv\_bfloat16} \) decimal exponential function in round-to-nearest mode.

Parameters
\( a \)
- \( \text{nv\_bfloat16} \). 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.

\[
\text{__device__ __nv\_bfloat16 hexp2 (const __nv\_bfloat16 a)}
\]
Calculates \( \text{nv\_bfloat16} \) binary exponential function in round-to-nearest mode.

Parameters
\( a \)
- \( \text{nv\_bfloat16} \). 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}
\hspace{1em} The
\hspace{1em} 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}
\hspace{1em} The
\hspace{1em} 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 \( \text{nv\_bf\_float16} \) decimal logarithm in round-to-nearest-even mode.

**Parameters**

\( \text{a} \)
- \( \text{nv\_bf\_float16} \). Is only being read.

**Returns**

\( \text{nv\_bf\_float16} \)
- The decimal logarithm of \( \text{a} \).

**Description**

Calculates \( \text{nv\_bf\_float16} \) decimal logarithm of input \( \text{a} \) in round-to-nearest-even mode.

__device__ __nv_bfloat16 hlog2 (const __nv_bfloat16 a)
Calculates \( \text{nv\_bf\_float16} \) binary logarithm in round-to-nearest-even mode.

**Parameters**

\( \text{a} \)
- \( \text{nv\_bf\_float16} \). Is only being read.

**Returns**

\( \text{nv\_bf\_float16} \)
- The binary logarithm of \( \text{a} \).

**Description**

Calculates \( \text{nv\_bf\_float16} \) binary logarithm of input \( \text{a} \) in round-to-nearest-even mode.

__device__ __nv_bfloat16 hrcp (const __nv_bfloat16 a)
Calculates \( \text{nv\_bf\_float16} \) reciprocal in round-to-nearest-even mode.

**Parameters**

\( \text{a} \)
- \( \text{nv\_bf\_float16} \). 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 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 floating-point format, with bfloat16way cases rounded to the nearest even integer value.

__device__ __nv_bfloat16 hsqrt (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 \( \texttt{nv\_bfloat16} \) reciprocal square root of input \( a \) in round-to-nearest mode.

\[
\texttt{__device__ \_\_nv\_bfloat16\ hsin (const \_\_nv\_bfloat16\ a)}
\]
Calculates \( \texttt{nv\_bfloat16} \) sine in round-to-nearest-even mode.

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

Returns
\( \texttt{nv\_bfloat16} \)

\[
\begin{align*}
\text{The} \\
\text{sine of} \ a.
\end{align*}
\]

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

\[
\texttt{__device__ \_\_nv\_bfloat16\ hsqrt (const \_\_nv\_bfloat16\ a)}
\]
Calculates \( \texttt{nv\_bfloat16} \) square root in round-to-nearest-even mode.

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

Returns
\( \texttt{nv\_bfloat16} \)

\[
\begin{align*}
\text{The} \\
\text{square root of} \ a.
\end{align*}
\]

Description
Calculates \( \texttt{nv\_bfloat16} \) square root of input \( a \) in round-to-nearest-even mode.
__device__ __nv_bfloat16 htrunc (const __nv_bfloat16 h)
Truncate input argument to the integral part.

Parameters
h
- nv_bfloat16. Is only being read.

Returns
nv_bfloat16
▷ The
truncated integer value.

Description
Round h to the nearest integer value that does not exceed h in magnitude.

1.2.7. 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
▶ The
elementwise cosine on vector a.

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
▶ The
elementwise exponential function on vector a.

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

__device__ __nv_bfloat162 h2exp10 (const __nv_bfloat162 a)
Calculates nv_bfloat162 vector decimal exponential function in round-to-nearest-even mode.

Parameters
a
- nv_bfloat162. Is only being read.
Returns
nv_bfloat162

- The elementwise decimal exponential function on vector $a$.

Description
Calculates $nv\_bfloat162$ decimal exponential function of input vector $a$ in round-to-nearest-even mode.

```c
__device__ __nv_bfloat162 h2exp2 (const __nv_bfloat162 a)
```
Calculates $nv\_bfloat162$ vector binary exponential function in round-to-nearest-even mode.

Parameters

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

Returns

nv_bfloat162

- The elementwise binary exponential function on vector $a$.

Description
Calculates $nv\_bfloat162$ binary exponential function of input vector $a$ in round-to-nearest-even mode.

```c
__device__ __nv_bfloat162 h2floor (const __nv_bfloat162 h)
```
Calculate the largest integer less than or equal to $h$.

Parameters

- $h$ - nv_bfloat162. Is only being read.

Returns

nv_bfloat162

- The vector of largest integers which is less than or equal to $h$. 

Description
For each component of vector \( h \) calculate the largest integer value which is less than or equal to \( h \).

```c
__device__ __nv_bfloat162 h2log (const __nv_bfloat162 a)
```
Calculates \( \text{nv\_bfloat162} \) vector natural logarithm in round-to-nearest-even mode.

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

Returns
\( \text{nv\_bfloat162} \)
- The elementwise natural logarithm on vector \( a \).

Description
Calculates \( \text{nv\_bfloat162} \) natural logarithm of input vector \( a \) in round-to-nearest-even mode.

```c
__device__ __nv_bfloat162 h2log10 (const __nv_bfloat162 a)
```
Calculates \( \text{nv\_bfloat162} \) vector decimal logarithm in round-to-nearest-even mode.

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

Returns
\( \text{nv\_bfloat162} \)
- The elementwise decimal logarithm on vector \( a \).

Description
Calculates \( \text{nv\_bfloat162} \) decimal logarithm of input vector \( a \) in round-to-nearest-even mode.
__device__ __nv_bfloat16 h2log2 (const __nv_bfloat16 a)
Calculates \texttt{nv\_bfloat16} vector binary logarithm in round-to-nearest-even mode.

Parameters
\begin{itemize}
\item \texttt{a}
  \begin{itemize}
  \item \texttt{nv\_bfloat16}. Is only being read.
  \end{itemize}
\end{itemize}

Returns
\texttt{nv\_bfloat16}

\begin{itemize}
\item The elementwise binary logarithm on vector \texttt{a}.
\end{itemize}

Description
Calculates \texttt{nv\_bfloat16} binary logarithm of input vector \texttt{a} in round-to-nearest mode.

__device__ __nv_bfloat16 h2rcp (const __nv_bfloat16 a)
Calculates \texttt{nv\_bfloat16} vector reciprocal in round-to-nearest-even mode.

Parameters
\begin{itemize}
\item \texttt{a}
  \begin{itemize}
  \item \texttt{nv\_bfloat16}. Is only being read.
  \end{itemize}
\end{itemize}

Returns
\texttt{nv\_bfloat16}

\begin{itemize}
\item The elementwise reciprocal on vector \texttt{a}.
\end{itemize}

Description
Calculates \texttt{nv\_bfloat16} reciprocal of input vector \texttt{a} in round-to-nearest-even mode.

__device__ __nv_bfloat16 h2rint (const __nv_bfloat16 h)
Round input to nearest integer value in \texttt{nv\_bfloat16} floating-point number.

Parameters
\begin{itemize}
\item \texttt{h}
  \begin{itemize}
  \item \texttt{nv\_bfloat16}. Is only being read.
  \end{itemize}
\end{itemize}
**Returns**
nv_bfloat162

- The vector of rounded integer values.

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

```
__device__ __nv_bfloat162 h2rsqrt (const __nv_bfloat162 a)
```
Calculates \( \text{nv\_bfloat162} \) vector reciprocal square root in round-to-nearest mode.

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

**Returns**
nv_bfloat162

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

**Description**
Calculates \( \text{nv\_bfloat162} \) reciprocal square root of input vector \( a \) in round-to-nearest-even mode.

```
__device__ __nv_bfloat162 h2sin (const __nv_bfloat162 a)
```
Calculates \( \text{nv\_bfloat162} \) vector sine in round-to-nearest-even mode.

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

**Returns**
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.

\begin{verbatim}
__device__ __nv_bfloat162 h2sqrt (const __nv_bfloat162 a)
\end{verbatim}

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}

The elementwise square root on vector \texttt{a}.

Description

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

\begin{verbatim}
__device__ __nv_bfloat162 h2trunc (const __nv_bfloat162 h)
\end{verbatim}

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}

The truncated \texttt{h}.

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 `rhypot()`, `cyl_bessel_i0()` are only available in device code.

Note that many floating-point and integer functions names are overloaded for different argument types. For example, the `log()` function has the following prototypes:

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

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.

```c
__device__ float acosf (float x)
```

Calculate the arc cosine of the input argument.

**Returns**

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

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

**Description**

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

**Note:**

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
__device__ float acoshf (float x)
Calculate the nonnegative arc hyperbolic cosine of the input argument.

Returns
Result will be in the interval [0, +∞].
- \(\text{acoshf}(1)\) 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 see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__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 see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
__device__ float asinhf (float x)
Calculate the arc hyperbolic sine of the input argument.

**Returns**
- asinhf(±0) returns ±0.
- asinhf(±∞) returns ±∞.

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

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

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

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

**Returns**
Result will be in radians, in the interval [−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 see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

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

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

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

Description
Calculate the cube root of x, \( x^{1/3} \).
__device__ float ceilf (float x)
Calculate ceiling of the input argument.

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

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

Description
Compute the smallest integer value not less than \(x\).

__device__ float copysignf (float x, float y)
Create value with given magnitude, copying sign of second value.

Returns
Returns a value with the magnitude of \(x\) and the sign of \(y\).

Description
Create a floating-point value with the magnitude \(x\) and the sign of \(y\).

__device__ float cosf (float x)
Calculate the cosine of the input argument.

Returns

- \(\text{cosf}(0)\) returns 1.
- \(\text{cosf}(\pm \infty)\) returns NaN.

Description
Calculate the cosine of the input argument \(x\) (measured in radians).
**__device__ float coshf (float x)**

Calculate the hyperbolic cosine of the input argument.

Returns

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

Description

Calculate the hyperbolic cosine of the input argument `x`.

**__device__ float cospif (float x)**

Calculate the cosine of the input argument `x × 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.
__device__ float cyl_bessel_i0f (float x)
Calculate the value of the regular modified cylindrical Bessel function of order 0 for the input argument.

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

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

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

__device__ float cyl_bessel_i1f (float x)
Calculate the value of the regular modified cylindrical Bessel function of order 1 for the input argument.

**Returns**
Returns the value of the regular modified cylindrical Bessel function of order 1.

**Description**
Calculate the value of the regular modified cylindrical Bessel function of order 1 for the input argument \( x \), \( I_1(x) \).

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

__device__ float erfcf (float x)
Calculate the complementary error function of the input argument.

**Returns**

- \( \text{erfcf( } - \infty \text{ ) returns 2.} \)
- \( \text{erfcf( } + \infty \text{ ) returns } +0. \)
Description
Calculate the complementary error function of the input argument \( x \), \( 1 - \text{erf}(x) \).

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

`__device__ float erfcinvf(float 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 see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

`__device__ float erfcxf(float x)`
Calculate the scaled complementary error function of the input argument.

Returns
- \( \text{erfcxf}(-\infty) \) returns +\( \infty \)
- \( \text{erfcxf}(+\infty) \) returns +0
- \( \text{erfcxf}(x) \) returns +0 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
- erff(±0) returns ±0.
- erff(±∞) returns ±1.

Description
Calculate the value of the error function for the input argument x, \[ \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt. \]

__device__ float erfinvf(float y)
Calculate the inverse error function of the input argument.

Returns
- erfinvf(1) returns +∞.
- erfinvf(-1) returns −∞.

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 = erf(x), for \(-1 \leq y \leq 1\), and \(-∞ \leq x \leq ∞\).
__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 see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
- This function is affected by the --use_fast_math compiler flag. See the CUDA C++ Programming Guide, Mathematical Functions Appendix, Intrinsic Functions section for a complete list of functions affected.

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

**Returns**
Returns $2^x$.

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

**Note:**
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
__device__ float expf (float x)
Calculate the base e exponential of the input argument.

Returns
Returns \( e^x \).

Description
Calculate the base e exponential of the input argument \( x \), \( e^x \).

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
- This function is affected by the --use_fast_math compiler flag. See the CUDA C++ Programming Guide, Mathematical Functions Appendix, Intrinsic Functions section for a complete list of functions affected.

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

Returns
Returns \( e^x - 1 \).

Description
Calculate the base e exponential of the input argument \( x \), minus 1.

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

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

Returns
Returns the absolute value of its argument.
- \( \text{fabs}(\pm \infty) \text{ returns } +\infty \).
- `fabs(±0)` returns 0.

**Description**

Calculate the absolute value of the input argument `x`.

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

### `__device__ float fdimf(float x, float y)`

**Description**

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 ≤ y`.

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

### `__device__ float fdividef(float x, float y)`

**Description**

Compute `x` divided by `y`. If `--use_fast_math` is specified, use `__fdividef()` for higher performance, otherwise use normal division.

**Returns**

Returns `x / y`.
\textbf{__device\_float floorf (float x)}

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

\textbf{Returns}

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

- \(\text{floorf}(\pm\infty)\) returns \(\pm\infty\).
- \(\text{floorf}(\pm0)\) returns \(\pm0\).

\textbf{Description}

Calculate the largest integer value which is less than or equal to \(x\).

\textbf{__device\_float fmaf (float x, float y, float z)}

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

\textbf{Returns}

Returns the rounded value of \(x \times y + z\) as a single operation.

- \(\text{fmaf}(\pm\infty, \pm0, z)\) returns NaN.
- \(\text{fmaf}(\pm0, \pm\infty, z)\) 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 see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float fmaxf (float x, float y)
Determine the maximum numeric value of the arguments.

Returns
Returns the maximum numeric values of the arguments $x$ and $y$.

- If both arguments are NaN, returns NaN.
- If one argument is NaN, returns the numeric argument.

Description
Determines the maximum numeric value of the arguments $x$ and $y$. Treats NaN arguments as missing data. If one argument is a NaN and the other is legitimate numeric value, the numeric value is chosen.

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

__device__ float fminf (float x, float y)
Determine the minimum numeric value of the arguments.

Returns
Returns the minimum numeric value of the arguments $x$ and $y$.

- If both arguments are NaN, returns NaN.
- If one argument is NaN, returns the numeric argument.
**Description**

Determines the minimum numeric value of the arguments \(x\) and \(y\). Treats NaN arguments as missing data. If one argument is a NaN and the other is legitimate numeric value, the numeric value is chosen.

**Note:**

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

```c
__device__ float fmodf (float x, float y)
```

Calculate the floating-point remainder of \(x / y\).

**Returns**

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

**Description**

Calculate the floating-point remainder of \(x / y\). The floating-point remainder of the division operation \(x / y\) calculated by this function is exactly the value \(x - n*y\), where \(n\) is \(x / y\) with its fractional part truncated. The computed value will have the same sign as \(x\), and its magnitude will be less than the magnitude of \(y\).

**Note:**

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

```c
__device__ float frexpf (float x, int *nptr)
```

Extract mantissa and exponent of a floating-point value.

**Returns**

Returns the fractional component \(m\).
- `frexp(0, nptr)` returns 0 for the fractional component and zero for the integer component.
- `frexp( ±0, nptr)` returns ±0 and stores zero in the location pointed to by nptr.
- `frexp( ±∞, nptr)` returns ±∞ and stores an unspecified value in the location to which nptr points.
- `frexp(NaN, y)` returns a NaN and stores an unspecified value in the location to which nptr points.

**Description**

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

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

```__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 see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
__device__ int ilogbf (float x)
Compute the unbiased integer exponent of the argument.

Returns
- If successful, returns the unbiased exponent of the argument.
- ilogbf(0) returns INT_MIN.
- ilogbf(NaN) returns INT_MIN.
- ilogbf(x) returns INT_MAX if x is $\infty$ or the correct value is greater than INT_MAX.
- 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.

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

__device__ __RETURN_TYPE isfinite (float a)
Determine whether argument is finite.

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

Description
Determine whether the floating-point value a is a finite value (zero, subnormal, or normal and not infinity or NaN).
__device__ __RETURN_TYPE isinf (float a)
Determine whether argument is infinite.

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

__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.

Description
Calculate the value of the Bessel function of the first kind of order 0 for the input argument x, $J_0(x)$. 
__device__ float j1f (float x)
Calculate the value of the Bessel function of the first kind of order 1 for the input argument.

Returns
Returns the value of the Bessel function of the first kind of order 1.

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

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

__device__ float jnf (int n, float x)
Calculate the value of the Bessel function of the first kind of order n for the input argument.

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

- jnf(n, NaN) returns NaN.
- jnf(n, x) returns NaN for n < 0.
- jnf(n, +∞) 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 \(x \cdot 2^{exp}\).

**Returns**

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

**Description**

Calculate the value of \(x \cdot 2^{exp}\) of the input arguments \(x\) and \(exp\).

---

**Note:**

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

**__device__ float lgammaf (float x)**

Calculate the natural logarithm of the absolute value of the gamma function of the input argument.

**Returns**

- lgammaf(1) returns +0.
- lgammaf(2) returns +0.
- lgammaf(x) returns \(\pm \infty\) if the correctly calculated value is outside the single floating-point range.
- lgammaf(x) returns \(+\infty\) if \(x \leq 0\) and \(x\) is an integer.
- lgammaf(-\(\infty\)) returns \(\infty\).
- lgammaf(\(+\infty\)) returns \(+\infty\).
Description

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

Note:

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

__device__ long long int llrintf (float x)

Round input to nearest integer value.

Returns

Returns rounded integer value.

Description

Round $x$ to the nearest integer value, with halfway cases rounded to the nearest even integer value. If the result is outside the range of the return type, the 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 llrintf().
__device__ float log10f (float x)
Calculate the base 10 logarithm of the input argument.

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

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

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
- This function is affected by the \(--use\_fast\_math\) compiler flag. See the CUDA C++ Programming Guide, Mathematical Functions Appendix, Intrinsic Functions section for a complete list of functions affected.

__device__ float log1pf (float x)
Calculate the value of \(\log_{e}(1 + x)\).

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

Description
Calculate the value of \(\log_{e}(1 + x)\) of the input argument \(x\).

Note:
__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 \(\pm \infty\).

Description

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

Note:

- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
- This function is affected by the \(--use_fast_math\) compiler flag. See the CUDA C++ Programming Guide, Mathematical Functions Appendix, Intrinsic Functions section for a complete list of functions affected.

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

Returns

- \( \logbf(\pm 0) \) returns \(- \infty\)
- \( \logbf(\pm \infty) \) returns \(\pm \infty\)

Description

Calculate the floating-point representation of the exponent of the input argument \( x \).

Note:
__device__ float logf(float x)
Calculate the natural logarithm of the input argument.

Returns

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

Description

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

Note:

- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
- This function is affected by the `--use_fast_math` compiler flag. See the CUDA C++ Programming Guide, Mathematical Functions Appendix, Intrinsic Functions section for a complete list of functions affected.

__device__ long int lrintf(float x)
Round input to nearest integer value.

Returns

Returns rounded integer value.

Description

Round $x$ to the nearest integer value, with halfway cases rounded to the nearest even integer value. If the result is outside the range of the return type, the 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 (const float a, const float b)
Calculate the maximum value of the input \texttt{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 (const float a, const float b)
Calculate the minimum value of the input \texttt{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 *iptr)
Break down the input argument into fractional and integral parts.

Returns
- \texttt{modff(\pm x, iptr)} returns a result with the same sign as \( x \).
- \texttt{modff(\pm \infty, iptr)} returns \( \pm 0 \) and stores \( \pm \infty \) in the object pointed to by \texttt{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 \( \text{iptr} \). Fractional and integral parts are given the same sign as the argument \( x \).

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

__device__ float nanf (const char *tagp)
Returns “Not a Number” value.

Returns
- nanf(tagp) returns NaN.

Description
Return a representation of a quiet NaN. Argument tagp selects one of the possible representations.

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

__device__ float nearbyintf (float x)
Round the input argument to the nearest integer.

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

Description
Round argument \( x \) to an integer value in single precision floating-point format. Uses round to nearest rounding, with ties rounding to even.
__device__ float nextafterf (float x, float y)

Return next representable single-precision floating-point value after argument $x$ in the direction of $y$.

Returns

- $\text{nextafterf}(x, y) = y$ if $x$ equals $y$
- $\text{nextafterf}(x, y) = \text{NaN}$ if either $x$ or $y$ are $\text{NaN}$

Description

Calculate the next representable single-precision floating-point value following $x$ in the direction of $y$. For example, if $y$ is greater than $x$, $\text{nextafterf()}$ returns the smallest representable number greater than $x$.

__device__ float norm3df (float a, float b, float c)

Calculate the square root of the sum of squares of three coordinates of the argument.

Returns

Returns the length of the 3D $\sqrt{p.x^2 + p.y^2 + p.z^2}$. 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.
__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.

Note:

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

__device__ float normcdff (float y)

Calculate the standard normal cumulative distribution function.

Returns

- $\text{normcdff}(+\infty)$ returns 1
- $\text{normcdff}(-\infty)$ returns +0

Description

Calculate the cumulative distribution function of the standard normal distribution for input argument $y$, $\Phi(y)$.

Note:

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
__device__ float normcdfinvf (float y)
Calculate the inverse of the standard normal cumulative distribution function.

Returns
- normcdfinvf(0) returns \(-\infty\).
- normcdfinvf(1) returns \(+\infty\).
- 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 see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__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 see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
__device__ float powf(float x, float y)

Calculate the value of first argument to the power of second argument.

Returns

- powf(±0, y) returns ±∞ for y an odd integer less than 0.
- powf(±0, y) returns +∞ for y less than 0 and not an odd integer.
- powf(±0, y) returns ±0 for y an odd integer greater than 0.
- powf(±0, y) returns +0 for y > 0 and not an odd integer.
- powf([-1, ±∞]) returns 1.
- powf(+1, y) returns 1 for any y, even a NaN.
- powf(x, ±0) returns 1 for any x, even a NaN.
- powf(x, y) returns a NaN for finite x < 0 and finite non-integer y.
- powf(x, −∞) returns +∞ for |x|< 1.
- powf(x, −∞) returns +0 for |x|> 1.
- powf(x, +∞) returns +0 for |x|< 1.
- powf(x, +∞) returns +∞ for |x|> 1.
- powf(−∞, y) returns -0 for y an odd integer less than 0.
- powf(−∞, y) returns +0 for y < 0 and not an odd integer.
- powf(−∞, y) returns −∞ for y an odd integer greater than 0.
- powf(−∞, y) returns +∞ for y > 0 and not an odd integer.
- powf(+∞, y) returns +0 for y < 0.
- powf(+∞, y) returns +∞ for y > 0.

Description

Calculate the value of x to the power of y.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
This function is affected by the `--use_fast_math` compiler flag. See the CUDA C++ Programming Guide, Mathematical Functions Appendix, Intrinsic Functions section for a complete list of functions affected.

### `__device__ float rcbrtf(float x)`
Calculate reciprocal cube root function.

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

**Description**
Calculate reciprocal cube root function of \( x \)

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

### `__device__ float remainderf(float x, float y)`
Compute single-precision floating-point remainder.

**Returns**
- \( remainderf(x, 0) \) returns NaN.
- \( remainderf(\pm \infty, y) \) returns NaN.
- \( remainderf(x, \pm \infty) \) returns \( x \) for finite \( x \).

**Description**
Compute single-precision floating-point remainder \( r \) of dividing \( x \) by \( y \) for nonzero \( y \). Thus \( r = x - ny \). The value \( n \) is the integer value nearest \( \frac{x}{y} \). In the case when \( |n - \frac{x}{y}| = \frac{1}{2} \), the even \( n \) value is chosen.

**Note:**
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
__device__ float remquof (float x, float y, int *quo)

Compute single-precision floating-point remainder and part of quotient.

**Returns**

Returns the remainder.

- remquof(x, 0, quo) returns NaN.
- 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.

**Note:**

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

__device__ float rhypotf (float x, float y)

Calculate one over the square root of the sum of squares of two arguments.

**Returns**

Returns one over the length of the hypotenuse \(\frac{1}{\sqrt{x^2 + y^2}}\). 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.

**Note:**

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
__device__ float rintf (float x)
Round input to nearest integer value in floating-point.

Returns
Returns rounded integer value.

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 +∞.

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

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

__device__ float rnorm4df (float a, float b, float c, float d)
Calculate one over the square root of the sum of squares of four coordinates of the argument.

Returns
Returns one over the length of the 3D vector \( \frac{1}{\sqrt{p.x^2 + p.y^2 + p.z^2 + p.w^2}} \). If the square root would overflow, returns 0. If the square root would underflow, returns +∞.
Description
Calculates one over the length of four dimension vector \( \mathbf{p} \) in Euclidean space without undue overflow or underflow.

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

\[ rnormf \text{ (int dim, const float *a)} \]
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 \( \mathbf{p} \), dimension of which is passed as an argument, in Euclidean space without undue overflow or underflow.

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

\[ \text{roundf (float x)} \]
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 \( \frac{1}{\sqrt{x}} \).

- `rsqrtf(+\infty)` returns \(+0\).
- `rsqrtf(0)` returns \(+\infty\).
- `rsqrtf(x)` returns NaN if \( x \) is less than 0.

Description
Calculate the reciprocal of the nonnegative square root of \( x \), \( \frac{1}{\sqrt{x}} \).

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

__device__ float scalblnf(float x, long int n)
Scale floating-point input by integer power of two.

Returns
Returns \( x \times 2^n \).

- `scalblnf(\pm0, n)` returns \( \pm0 \).
- `scalblnf(x, 0)` returns \( x \).
- `scalblnf(\pm\infty, n)` returns \( \pm\infty \).

Description
Scale \( x \) by \( 2^n \) by efficient manipulation of the floating-point exponent.
**__device__ float scalbnf (float x, int n)**
Scale floating-point input by integer power of two.

**Returns**
Returns \(x \times 2^n\).

- `scalbnf(\pm0, n)` returns \(\pm0\).
- `scalbnf(x, 0)` returns \(x\).
- `scalbnf(\pm\infty, n)` returns \(\pm\infty\).

**Description**
Scale \(x\) by \(2^n\) by efficient manipulation of the floating-point exponent.

**__device__ RETURN_TYPE signbit (float a)**
Return the sign bit of the input.

**Returns**
Reports the sign bit of all values including infinities, zeros, and NaNs.

- With Visual Studio 2013 host compiler: `RETURN_TYPE` is `bool`. Returns true if and only if \(a\) is negative.
- With other host compilers: `RETURN_TYPE` is `int`. Returns a nonzero value if and only if \(a\) is negative.

**Description**
Determine whether the floating-point value \(a\) is negative.

**__device__ void sincosf (float x, float *sptr, float *cptr)**
Calculate the sine and cosine of the first input argument.

**Returns**

- none

**Description**
Calculate the sine and cosine of the first input argument \(x\) (measured in radians). The results for sine and cosine are written into the second argument, `sptr`, and, respectively, third argument, `cptr`. 
See also:
sinf() and cosf().

Note:
▶ For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
▶ This function is affected by the --use_fast_math compiler flag. See the CUDA C++ Programming Guide, Mathematical Functions Appendix, Intrinsic Functions section for a complete list of functions affected.

__device__ void sincospif (float x, float *sptr, float *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:
sinpif() and cospif().

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

__device__ float sinf (float x)
Calculate the sine of the input argument.

Returns
▶ sinf( ±0 ) returns ±0.
▶ sinf( ±∞ ) returns NaN.
Description
Calculate the sine of the input argument $x$ (measured in radians).

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
- This function is affected by the `--use_fast_math` compiler flag. See the CUDA C++ Programming Guide, Mathematical Functions Appendix, Intrinsic Functions section for a complete list of functions affected.

```__device__ float sinhf (float x)```
Calculate the hyperbolic sine of the input argument.

Returns
- $\sinhf(\pm 0)$ returns $\pm 0$.
- $\sinhf(\pm \infty)$ returns $\pm \infty$.

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

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

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

Returns
- $\sinpif(\pm 0)$ returns $\pm 0$.
- $\sinpif(\pm \infty)$ returns NaN.

Description
Calculate the sine of $x \times \pi$ [measured in radians], where $x$ is the input argument.
__device__ float sqrtf (float x)
Calculate the square root of the input argument.

Returns
Returns $\sqrt{x}$.

- $\sqrt{0}$ returns $\pm 0$.
- $\sqrt{\pm \infty}$ returns $+\infty$.
- $\sqrt{x}$ returns NaN if $x$ is less than 0.

Description
Calculate the nonnegative square root of $x$, $\sqrt{x}$.

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

Returns

- $\tan(0)$ returns $\pm 0$.
- $\tan(\pm \infty)$ returns NaN.

Description
Calculate the tangent of the input argument $x$ (measured in radians).
This function is affected by the --use_fast_math compiler flag. See the CUDA C++ Programming Guide, Mathematical Functions Appendix, Intrinsic Functions section for a complete list of functions affected.

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

Returns
- tanhf(±0) returns ±0.

Description
Calculate the hyperbolic tangent of the input argument x.

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

__device__ float tgammaf (float x)
Calculate the gamma function of the input argument.

Returns
- tgammaf(±0) returns ±∞.
- tgammaf(2) returns +1.
- tgammaf(x) returns ±∞ if the correctly calculated value is outside the single floating-point range.
- tgammaf(x) returns NaN if x < 0 and x is an integer.
- tgammaf(−∞) returns NaN.
- tgammaf(+) returns +∞.

Description
Calculate the gamma function of the input argument x, namely the value of \( \int_0^\infty e^{-t^{x-1}} dt \).
__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(NaN) returns NaN.

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

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

__device__ float y1f (float x)
Calculate the value of the Bessel function of the second kind of order 1 for the input argument.

Returns
Returns the value of the Bessel function of the second kind of order 1.
y1f(0) returns $-\infty$.

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 $x$, $Y_1(x)$.

**Note:**

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

```c
__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 $x$.

**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 see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
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 see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

```c
__device__ double acosh (double x)
```
Calculate the nonnegative arc hyperbolic cosine of the input argument.

Returns

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

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

Description

Calculate the nonnegative arc hyperbolic cosine of the input argument $x$.

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

__device__ double asinh (double x)
Calculate the arc hyperbolic sine of the input argument.

Returns

- asinh(\(\pm 0\)) returns \(\pm 0\).
- asinh(\(\pm \infty\)) returns \(\pm \infty\).

Description
Calculate the arc hyperbolic sine of the input argument \(x\).

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

__device__ double atan2 (double y, double x)
Calculate the arc tangent of the ratio of first and second input arguments.

Returns
Result will be in radians, in the interval [- $\pi$, + $\pi$].
- 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 see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

__device__ double atanh (double x)
Calculate the arc hyperbolic tangent of the input argument.

Returns
- atanh(±0) returns ±0.
- \( \text{atanh} (\pm 1) \) returns \( \pm \infty \).
- \( \text{atanh}(x) \) returns NaN for \( x \) outside interval \([-1, 1]\).

**Description**

Calculate the arc hyperbolic tangent of the input argument \( x \).

**Note:**

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

```c
__device__ double cbrt (double x)
```

Calculate the cube root of the input argument.

**Returns**

Returns \( x^{1/3} \).

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

**Description**

Calculate the cube root of \( x \), \( x^{1/3} \).

**Note:**

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

```c
__device__ double ceil (double x)
```

Calculate ceiling of the input argument.

**Returns**

Returns \( \lceil x \rceil \) expressed as a floating-point number.

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

**Description**

Compute the smallest integer value not less than \( x \).
__device__ double copysign (double x, double y)
Create value with given magnitude, copying sign of second value.

Returns
Returns a value with the magnitude of \(x\) and the sign of \(y\).

Description
Create a floating-point value with the magnitude \(x\) and the sign of \(y\).

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

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

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

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

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

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

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

Note:
__device__ double cospi (double x)
Calculate the cosine of the input argument $x \times \pi$.

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

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

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

__device__ double cyl_bessel_i0 (double x)
Calculate the value of the regular modified cylindrical Bessel function of order 0 for the input argument.

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

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

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
__device__ double cyl_bessel_i1 (double x)
Calculate the value of the regular modified cylindrical Bessel function of order 1 for the input argument.

Returns
Returns the value of the regular modified cylindrical Bessel function of order 1.

Description
Calculate the value of the regular modified cylindrical Bessel function of order 1 for the input argument \(x\), \(I_1(x)\).

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

__device__ double erf (double x)
Calculate the error function of the input argument.

Returns
- \(\text{erf}(\pm 0)\) returns \(\pm 0\).
- \(\text{erf}(\pm \infty)\) returns \(\pm 1\).

Description
Calculate the value of the error function for the input argument \(x\), \(\frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt\).

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
__device__ double erfc (double x)
Calculate the complementary error function of the input argument.

Returns
- \( \text{erfc}( -\infty ) \) returns 2.
- \( \text{erfc}( +\infty ) \) returns +0.

Description
Calculate the complementary error function of the input argument \( x \), \( 1 - \text{erf}(x) \).

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

__device__ double erfcinv (double 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 see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
__device__ double erfcx (double x)
Calculate the scaled complementary error function of the input argument.

Returns
- erfcx\(-\infty\) returns +\infty
- 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 see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

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

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

**Returns**
Returns $10^x$.

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

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

__device__ double exp2 (double x)
Calculate the base 2 exponential of the input argument.

**Returns**
Returns $2^x$.

**Description**
Calculate the base 2 exponential of the input argument $x$. 
Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

__device__ double expm1 (double x)
Calculate the base $e$ exponential of the input argument, minus 1.

Returns
Returns $e^x - 1$.

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

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

__device__ double fabs (double x)
Calculate the absolute value of the input argument.

Returns
Returns the absolute value of the input argument.

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

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

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
__device__ double fdim (double x, double y)

Compute the positive difference between \( x \) and \( y \).

Returns

Returns the positive difference between \( x \) and \( y \).

- \( \text{fdim}(x, y) \) returns \( x - y \) if \( x > y \).
- \( \text{fdim}(x, y) \) returns \( +0 \) if \( x \leq y \).

Description

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

Note:

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

__device__ double floor (double x)

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

Returns

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

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

Description

Calculates the largest integer value which is less than or equal to \( x \).

Note:

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
__device__ double fma (double x, double y, double z)
Compute \(x \times y + z\) as a single operation.

Returns
Returns the rounded value of \(x \times y + z\) as a single operation.

- \(fma(\pm\infty, \pm 0, z)\) 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
Compute the value of \(x \times y + z\) as a single ternary operation. After computing the value to infinite precision, the value is rounded once.

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

__device__ double fmax (double, double)
Determine the maximum numeric value of the arguments.

Returns
Returns the maximum numeric values of the arguments \(x\) and \(y\).

- If both arguments are NaN, returns NaN.
- If one argument is NaN, returns the numeric argument.

Description
Determines the maximum numeric value of the arguments \(x\) and \(y\). Treats NaN arguments as missing data. If one argument is a NaN and the other is legitimate numeric value, the numeric value is chosen.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
__device__ double fmin (double x, double y)
Determine the minimum numeric value of the arguments.

Returns
Returns the minimum numeric value of the arguments x and y.
- If both arguments are NaN, returns NaN.
- If one argument is NaN, returns the numeric argument.

Description
Determines the minimum numeric value of the arguments x and y. Treats NaN arguments as missing data. If one argument is a NaN and the other is legitimate numeric value, the numeric value is chosen.

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

__device__ double fmod (double x, double y)
Calculate the double-precision floating-point remainder of x / y.

Returns
- Returns the floating-point remainder of x / y.
- fmod( ±0 , y) returns ±0 if y is not zero.
- fmod(x, ±∞ ) returns x if x is finite.
- fmod(x, y) returns NaN if x is ±∞ or y is zero.
- If either argument is NaN, NaN is returned.

Description
Calculate the double-precision floating-point remainder of x / y. The floating-point remainder of the division operation x / y calculated by this function is exactly the value x − n*y, where n is x / y with its fractional part truncated. The computed value will have the same sign as x, and its magnitude will be less than the magnitude of y.

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

__device__ double hypot (double x, double y)

Calculate the square root of the sum of squares of two arguments.

**Returns**

Returns the length of the hypotenuse \( \sqrt{x^2 + y^2} \). 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.
__device__ int ilogb (double x)
Compute the unbiased integer exponent of the argument.

Returns
- If successful, returns the unbiased exponent of the argument.
- ilogb(0) returns INT_MIN.
- ilogb(NaN) returns INT_MIN.
- ilogb(x) returns INT_MAX if x is \(\infty\) or the correct value is greater than INT_MAX.
- ilogb(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 (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).
\texttt{\_\_device\_\_ \_RETURN\_TYPE isinf (double a)}

Determine whether argument is infinite.

\textbf{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.

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

\texttt{\_\_device\_\_ \_RETURN\_TYPE isnan (double a)}

Determine whether argument is a NaN.

\textbf{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.

\textbf{Description}
Determine whether the floating-point value \( a \) is a NaN.

\texttt{\_\_device\_\_ double j0 (double x)}

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

\textbf{Returns}
Returns the value of the Bessel function of the first kind of order 0.
- \( j_0(\pm \infty) \) returns +0.
- \( j_0(\text{NaN}) \) returns NaN.

\textbf{Description}
Calculate the value of the Bessel function of the first kind of order 0 for the input argument \( x \), \( J_0(x) \).
__device__ double j1 (double x)
Calculate the value of the Bessel function of the first kind of order 1 for the input argument.

Returns
Returns the value of the Bessel function of the first kind of order 1.
- j1(±0) returns ±0.
- j1(±∞) returns ±0.
- j1(NaN) returns NaN.

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

__device__ double jn (int n, double x)
Calculate the value of the Bessel function of the first kind of order \( n \) for the input argument.

Returns
Returns the value of the Bessel function of the first kind of order \( n \).
- jn\([n, NaN]\) returns NaN.
- jn\([n, x]\) returns NaN for \( n < 0 \).
- jn\([n, +\infty]\) returns +0.

Description
Calculate the value of the Bessel function of the first kind of order \( n \) for the input argument \( x \), \( \text{\(J_n\)}(\text{\(x\)}) \).
\texttt{__device__ double ldexp (double x, int exp)}

Calculate the value of $x \cdot 2^{exp}$.

Returns

- $\text{ldexp}(x)$ returns $\pm \infty$ if the correctly calculated value is outside the double floating-point range.

Description

Calculate the value of $x \cdot 2^{exp}$ of the input arguments $x$ and $exp$.

\texttt{__device__ double lgamma (double x)}

Calculate the natural logarithm of the absolute value of the gamma function of the input argument.

Returns

- $\text{lgamma}(1)$ returns $+0$.
- $\text{lgamma}(2)$ returns $+0$.
- $\text{lgamma}(x)$ returns $\pm \infty$ if the correctly calculated value is outside the double floating-point range.
- $\text{lgamma}(x)$ returns $+\infty$ if $x \leq 0$ and $x$ is an integer.
- $\text{lgamma}(+\infty)$ returns $+\infty$.
- $\text{lgamma}(+\infty)$ returns $+\infty$. 
**Description**

Calculate the natural logarithm of the absolute value of the gamma function of the input argument $x$, namely the value of $\log_e \left| \int_0^\infty e^{-t} t^{x-1} dt \right|$

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

```c
__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.

```c
__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 `llrint()`.
__device__ double log (double x)

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

Returns

- $\log(\pm0)$ returns $-\infty$.
- $\log(1)$ returns $+0$.
- $\log(x)$ returns NaN for $x < 0$.
- $\log(+\infty)$ returns $+\infty$

Description

Calculate the base $e$ logarithm of the input argument $x$.

Note:

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

__device__ double log10 (double x)

Calculate the base 10 logarithm of the input argument.

Returns

- $\log_{10}(\pm0)$ returns $-\infty$.
- $\log_{10}(1)$ returns $+0$.
- $\log_{10}(x)$ returns NaN for $x < 0$.
- $\log_{10}(+\infty)$ returns $+\infty$.

Description

Calculate the base 10 logarithm of the input argument $x$.

Note:

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
__device__ double log1p (double x)
Calculate the value of $\log_e(1 + x)$.

Returns

- log1p(±0) returns ±0.
- log1p(-1) returns $-\infty$.
- log1p(x) returns NaN for $x < -1$.
- log1p(+∞) returns +∞.

Description

Calculate the value of $\log_e(1 + x)$ of the input argument x.

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

__device__ double log2 (double x)
Calculate the base 2 logarithm of the input argument.

Returns

- log2(±0) returns $-\infty$.
- log2(1) returns +0.
- log2(x) returns NaN for $x < 0$.
- log2(+∞) returns +∞.

Description

Calculate the base 2 logarithm of the input argument x.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
__device__ double logb (double x)
Calculate the floating-point representation of the exponent of the input argument.

Returns

- logb ±0 returns \(-\infty\)
- logb ±\(\infty\) returns \(+\infty\)

Description
Calculate the floating-point representation of the exponent of the input argument \(x\).

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

__device__ long int lrint (double x)
Round input to nearest integer value.

Returns
Returns rounded integer value.

Description
Round \(x\) to the nearest integer value, with halfway cases rounded to the nearest even integer value. If the result is outside the range of the return type, the 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.
This function may be slower than alternate rounding methods. See [rint](#).

```cpp
__device__ double max (const double a, const float b)
```

Calculate the maximum value of the input `double` and `float` arguments.

**Description**

Convert `float` argument `b` to `double`, followed by `fmax`.

Note, this is different from `std::` specification.

```cpp
__device__ double max (const float a, const double b)
```

Calculate the maximum value of the input `float` and `double` arguments.

**Description**

Convert `float` argument `a` to `double`, followed by `fmax`.

Note, this is different from `std::` specification.

```cpp
__device__ double max (const double a, const double b)
```

Calculate the maximum value of the input `float` arguments.

**Description**

Calculate the maximum value of the arguments `a` and `b`. Behavior is equivalent to `fmax` function.

Note, this is different from `std::` specification.

```cpp
__device__ double min (const double a, const float b)
```

Calculate the minimum value of the input `double` and `float` arguments.

**Description**

Convert `float` argument `b` to `double`, followed by `fmin`.

Note, this is different from `std::` specification.
__device__ double min (const float a, const double b)
Calculate the minimum value of the input float and double arguments.

Description
Convert float argument a to double, followed by fmin().
Note, this is different from std:: specification

__device__ double min (const double a, const double b)
Calculate the minimum value of the input float arguments.

Description
Calculate the minimum value of the arguments a and b. Behavior is equivalent to fmin() function.
Note, this is different from std:: specification

__device__ double modf (double x, double *iptr)
Break down the input argument into fractional and integral parts.

Returns
- modf(±x, iptr) returns a result with the same sign as x.
- modf(±∞, iptr) returns ±0 and stores ±∞ in the object pointed to by iptr.
- modf(NaN, iptr) stores a NaN in the object pointed to by iptr and returns a NaN.

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

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

__device__ double nearbyint (double x)
Round the input argument to the nearest integer.

Returns

- nearbyint(±0) returns ±0.
- nearbyint(±∞) returns ±∞.

Description
Round argument x to an integer value in double precision floating-point format. Uses round to nearest rounding, with ties rounding to even.

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

__device__ double nextafter (double x, double y)
Return next representable double-precision floating-point value after argument x in the direction of y.

Returns

- nextafter(x, y) = y if x equals y
nextafter\(x, y\) = NaN if either \(x\) or \(y\) are NaN

**Description**

Calculate the next representable double-precision floating-point value following \(x\) in the direction of \(y\). For example, if \(y\) is greater than \(x\), `nextafter()` returns the smallest representable number greater than \(x\).

**Note:**

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

__device__ double norm (int dim, const double *t)

Calculate the square root of the sum of squares of any number of coordinates.

**Returns**

Returns the length of the dim-D vector \(\sqrt{p.1^2 + p.2^2 + \ldots + p.dim^2}\). 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 see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

__device__ double norm3d (double a, double b, double c)

Calculate the square root of the sum of squares of three coordinates of the argument.

**Returns**

Returns the length of 3D vector \(\sqrt{p.x^2 + p.y^2 + p.z^2}\). 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 see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

__device__ double norm4d (double a, double b, double c, double d)
Calculate the square root of the sum of squares of four coordinates of the argument.

Returns
Returns the length of 4D vector \( \sqrt{a^2 + b^2 + c^2 + d^2} \). 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 see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

__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( ±0 , y) returns ±∞ for y an odd integer less than 0.
- pow( ±0 , y) returns $+\infty$ for y less than 0 and not an odd integer.
- pow( ±0 , y) returns 0 for y an odd integer greater than 0.
- pow( ±0 , y) returns +0 for y > 0 and not an odd integer.
- pow(-1, ±∞ ) returns 1.
- pow(+1, y) returns 1 for any y, even a NaN.
- pow|x|, ±0 ) returns 1 for any x, even a NaN.
- pow(|x|, y) returns a NaN for finite x < 0 and finite non-integer y.
• pow(x, −∞) returns +∞ for |x| < 1.
• pow(x, −∞) returns +0 for |x| > 1.
• pow(x, +∞) returns +0 for |x| < 1.
• pow(x, +∞) returns +∞ for |x| > 1.
• powl(−∞, y) returns -0 for y an odd integer less than 0.
• powl(−∞, y) returns +0 for y < 0 and not an odd integer.
• powl(−∞, y) returns −∞ for y an odd integer greater than 0.
• powl(−∞, y) returns +∞ for y > 0 and not an odd integer.
• powl(+∞, y) returns +0 for y < 0.
• powl(+∞, y) returns +∞ for y > 0.

Description
Calculate the value of x to the power of y

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

__device__ double 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 see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
__device__ double remainder (double x, double y)
Compute double-precision floating-point remainder.

**Returns**

- remainder(x, 0) returns NaN.
- remainder(±∞, y) returns NaN.
- remainder(x, ±∞) returns x for finite x.

**Description**

Compute double-precision floating-point remainder \( r \) of dividing \( x \) by \( y \) for nonzero \( y \). Thus \( r = x - ny \). The value \( n \) is the integer value nearest \( \frac{x}{y} \). In the case when \( |n - \frac{x}{y}| = \frac{1}{2} \), the even \( n \) value is chosen.

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

__device__ double remquo (double x, double y, int *quo)
Compute double-precision floating-point remainder and part of quotient.

**Returns**

Returns the remainder.

- remquo(x, 0, quo) returns NaN.
- remquo(±∞, y, quo) returns NaN.
- remquo(x, ±∞, 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.
__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.

__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.

---

**Note:**

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

---

**__device__ double rnorm3d (double a, double b, double c)\**

Calculate one over the square root of the sum of squares of three coordinates 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**

Calculate one over the length of three dimensional vector \( p \) in Euclidean space undue overflow or underflow.

---

**Note:**

For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
__device__ double rnorm4d (double a, double b, double c, double d)
Calculate one over the square root of the sum of squares of four coordinates of the argument.

Returns
Returns one over the length of the 3D vector \( \frac{1}{\sqrt{p.x^2 + p.y^2 + p.z^2 + p.t^2}} \). 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.

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

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

Returns
Returns rounded integer value.

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 rint().

__device__ double rsqrt (double x)
Calculate the reciprocal of the square root of the input argument.

Returns
Returns \( \frac{1}{\sqrt{x}} \).
- \( \text{rsqrt} (+\infty) \) returns +0.
- \( \text{rsqrt} (\pm 0) \) returns \( \pm \infty \).
- \( \text{rsqrt}(x) \) returns NaN if \( x \) is less than 0.

**Description**

Calculate the reciprocal of the nonnegative square root of \( x \), \( \frac{1}{\sqrt{x}} \).

**Note:**

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

**\text{__device__ double scalbln (double x, long int n)}**

Scale floating-point input by integer power of two.

**Returns**

Returns \( x \times 2^n \).
- \( \text{scalbln}(\pm 0, n) \) returns \( \pm 0 \).
- \( \text{scalbln}(x, 0) \) returns \( x \).
- \( \text{scalbln}(\pm \infty, n) \) returns \( \pm \infty \).

**Description**

Scale \( x \) by \( 2^n \) by efficient manipulation of the floating-point exponent.

**\text{__device__ double scalbn (double x, int n)}**

Scale floating-point input by integer power of two.

**Returns**

Returns \( x \times 2^n \).
- \( \text{scalbn}(\pm 0, n) \) returns \( \pm 0 \).
- \( \text{scalbn}(x, 0) \) returns \( x \).
- \( \text{scalbn}(\pm \infty, n) \) returns \( \pm \infty \).

**Description**

Scale \( x \) by \( 2^n \) by efficient manipulation of the floating-point exponent.
__device__ __RETURN_TYPE signbit (double a)
Return the sign bit of the input.

Returns
Reports the sign bit of all values including infinities, zeros, and NaNs.

‣ With Visual Studio 2013 host compiler: __RETURN_TYPE is 'bool'. Returns true if and only if a is negative.
‣ With other host compilers: __RETURN_TYPE is 'int'. Returns a nonzero value if and only if a is negative.

Description
Determine whether the floating-point value a is negative.

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

__device__ void sincos (double x, double *sptr, double *cptr)
Calculate the sine and cosine of the first input argument.

Returns
‣ none
Description
Calculate the sine and cosine of the first input argument \( x \) (measured in radians). The results for sine and cosine are written into the second argument, \( \text{spt}_r \), and, respectively, third argument, \( \text{cptr} \).

See also:
sin() and cos().

\[
\text{__device__ void sincospi (double x, double *sptr, double *cptr)}
\]
Calculate the sine and cosine of the first input argument \( x \pi \).

Returns
- none

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

See also:
sinpi() and cospi().

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
__device__ double sinh (double x)
Calculate the hyperbolic sine of the input argument.

Returns
- \( \sinh(\pm 0) \) returns \( \pm 0 \).
- \( \sinh(\pm \infty) \) returns \( \pm \infty \).

Description
Calculate the hyperbolic sine of the input argument \( x \).

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

__device__ double sinpi (double x)
Calculate the sine of the input argument \( x \pi \).

Returns
- \( \sin pi(\pm 0) \) returns \( \pm 0 \).
- \( \sin pi(\pm \infty) \) returns NaN.

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

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

__device__ double sqrt (double x)
Calculate the square root of the input argument.

Returns
- Returns \( \sqrt{x} \).
- \( \sqrt{\pm 0} \) returns \( \pm 0 \).
- $\sqrt{+\infty}$ returns $+\infty$.
- $\sqrt{x}$ returns NaN if $x$ is less than 0.

**Description**

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

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

### __device__ double tan (double x)

Calculate the tangent of the input argument.

**Returns**
- $\tan(\pm 0)$ returns $\pm 0$.
- $\tan(\pm \infty)$ returns NaN.

**Description**

Calculate the tangent of the input argument $x$ (measured in radians).

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

### __device__ double tanh (double x)

Calculate the hyperbolic tangent of the input argument.

**Returns**
- $\tanh(\pm 0)$ returns $\pm 0$.

**Description**

Calculate the hyperbolic tangent of the input argument $x$.

**Note:**
__device__ double tgamma (double x)
Calculate the gamma function of the input argument.

Returns

- \( \text{tgamma}(-0) \) 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.
- \( \text{tgamma}(\infty) \) 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 see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

__device__ double trunc (double x)
Truncate input argument to the integral part.

Returns

Returns truncated integer value.

Description

Round \( x \) to the nearest integer value that does not exceed \( x \) in magnitude.
__device__ double y0 (double x)
Calculate the value of the Bessel function of the second kind of order 0 for the input argument.

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

- y0(0) returns $-\infty$.
- y0(x) returns NaN for $x < 0$.
- y0(+\infty) returns +0.
- y0(NaN) returns NaN.

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

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

__device__ double y1 (double x)
Calculate the value of the Bessel function of the second kind of order 1 for the input argument.

Returns
Returns the value of the Bessel function of the second kind of order 1.

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

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

Note:
__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 -∞.
- yn(n, x) returns NaN for x < 0.
- yn(n, +∞) returns +0.
- yn(n, NaN) returns NaN.

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

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

1.6. 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 (const long long int a, const long long int b)
Calculate the maximum value of the input long long int arguments.

Description
Calculate the maximum value of the arguments a and b.

__device__ long long int llmin (const long long int a, const long long int b)
Calculate the minimum value of the input long long int arguments.

Description
Calculate the minimum value of the arguments a and b.

__device__ unsigned long long int max (const unsigned long long int a, const long long int b)
Calculate the maximum value of the input unsigned long long int and long long int arguments.

Description
Calculate the maximum value of the arguments a and b, perform integer promotion first.
__device__ unsigned long long int max (const long long int a, const unsigned long long int b)
Calculate the maximum value of the input long long int and unsigned long long int arguments.

Description
Calculate the maximum value of the arguments a and b, perform integer promotion first.

__device__ unsigned long long int max (const unsigned long long int a, const unsigned long long int b)
Calculate the maximum value of the input unsigned long long int arguments.

Description
Calculate the maximum value of the arguments a and b.

__device__ long long int max (const long long int a, const long long int b)
Calculate the maximum value of the input long long int arguments.

Description
Calculate the maximum value of the arguments a and b.

__device__ unsigned long int max (const unsigned long int a, const long int b)
Calculate the maximum value of the input unsigned long int and long int arguments.

Description
Calculate the maximum value of the arguments a and b, perform integer promotion first.
__device__ unsigned long int max (const long int a, const unsigned long int b)
Calculate the maximum value of the input long int and unsigned long int arguments.

Description
Calculate the maximum value of the arguments a and b, perform integer promotion first.

__device__ unsigned long int max (const unsigned long int a, const unsigned long int b)
Calculate the maximum value of the input unsigned long int arguments.

Description
Calculate the maximum value of the arguments a and b.

__device__ long int max (const long int a, const long int b)
Calculate the maximum value of the input long int arguments.

Description
Calculate the maximum value of the arguments a and b.

__device__ unsigned int max (const unsigned int a, const int b)
Calculate the maximum value of the input unsigned int and int arguments.

Description
Calculate the maximum value of the arguments a and b, perform integer promotion first.

__device__ unsigned int max (const int a, const unsigned int b)
Calculate the maximum value of the input int and unsigned int arguments.

Description
Calculate the maximum value of the arguments a and b, perform integer promotion first.
__device__ unsigned int max (const unsigned int a, const unsigned int b)
Calculate the maximum value of the input unsigned int arguments.

Description
Calculate the maximum value of the arguments a and b.

__device__ int max (const int a, const int b)
Calculate the maximum value of the input int arguments.

Description
Calculate the maximum value of the arguments a and b.

__device__ unsigned long long int min (const unsigned long long int a, const long long int b)
Calculate the minimum value of the input unsigned long long int and long long int arguments.

Description
Calculate the minimum value of the arguments a and b, perform integer promotion first.

__device__ unsigned long long int min (const long long int a, const unsigned long long int b)
Calculate the minimum value of the input long long int and unsigned long long int arguments.

Description
Calculate the minimum value of the arguments a and b, perform integer promotion first.

__device__ unsigned long long int min (const unsigned long long int a, const unsigned long long int b)
Calculate the minimum value of the input unsigned long long int arguments.

Description
Calculate the minimum value of the arguments a and b.
__device__ long long int min (const long long int a, const long long int b)
Calculate the minimum value of the input long long int arguments.

Description
Calculate the minimum value of the arguments a and b.

__device__ unsigned long int min (const unsigned long int a, const long int b)
Calculate the minimum value of the input unsigned long int and long int arguments.

Description
Calculate the minimum value of the arguments a and b, perform integer promotion first.

__device__ unsigned long int min (const long int a, const unsigned long int b)
Calculate the minimum value of the input long int and unsigned long int arguments.

Description
Calculate the minimum value of the arguments a and b, perform integer promotion first.

__device__ unsigned long int min (const unsigned long int a, const unsigned long int b)
Calculate the minimum value of the input unsigned long int arguments.

Description
Calculate the minimum value of the arguments a and b.

__device__ long int min (const long int a, const long int b)
Calculate the minimum value of the input long int arguments.

Description
Calculate the minimum value of the arguments a and b.
__device__ unsigned int min (const unsigned int a, const int b)
Calculate the minimum value of the input unsigned int and int arguments.

Description
Calculate the minimum value of the arguments a and b, perform integer promotion first.

__device__ unsigned int min (const int a, const unsigned int b)
Calculate the minimum value of the input int and unsigned int arguments.

Description
Calculate the minimum value of the arguments a and b, perform integer promotion first.

__device__ unsigned int min (const unsigned int a, const unsigned int b)
Calculate the minimum value of the input unsigned int arguments.

Description
Calculate the minimum value of the arguments a and b.

__device__ int min (const int a, const int b)
Calculate the minimum value of the input int arguments.

Description
Calculate the minimum value of the arguments a and b.

__device__ unsigned long long int ullmax (const unsigned long long int a, const unsigned long long int b)
Calculate the maximum value of the input unsigned long long int arguments.

Description
Calculate the maximum value of the arguments a and b.
__device__ unsigned long long int ullmin (const unsigned long long int a, const unsigned long long int b)
Calculate the minimum value of the input unsigned long long int arguments.

Description
Calculate the minimum value of the arguments a and b.

__device__ unsigned int umax (const unsigned int a, const unsigned int b)
Calculate the maximum value of the input unsigned int arguments.

Description
Calculate the maximum value of the arguments a and b.

__device__ unsigned int umin (const unsigned int a, const unsigned int b)
Calculate the minimum value of the input unsigned int arguments.

Description
Calculate the minimum value of the arguments a and b.

1.7. 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.
__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 $x$, $10^x$.

__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 $x$, $e^x$.

__device__ float __fadd_rd (float x, float y)
Add two floating-point values in round-down mode.

Returns
Returns $x + y$. 
Description
Compute the sum of \( x \) and \( y \) in round-down (to negative infinity) mode.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.

__device__ float __fadd_rn (float x, float y)
Add two floating-point values in round-to-nearest-even mode.

Returns
Returns \( x + y \).

Description
Compute the sum of \( x \) and \( y \) in round-to-nearest-even rounding mode.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.

__device__ float __fadd_ru (float x, float y)
Add two floating-point values in round-up mode.

Returns
Returns \( x + y \).

Description
Compute the sum of \( x \) and \( y \) in round-up (to positive infinity) mode.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
__device__ float __fadd_rz (float x, float y)
Add two floating-point values in round-towards-zero mode.

Returns
Returns $x + y$.

Description
Compute the sum of $x$ and $y$ in round-towards-zero mode.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.

__device__ float __fdiv_rd (float x, float y)
Divide two floating-point values in round-down mode.

Returns
Returns $x / y$.

Description
Divide two floating-point values $x$ by $y$ in round-down (to negative infinity) mode.

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

__device__ float __fdiv_rn (float x, float y)
Divide two floating-point values in round-to-nearest-even mode.

Returns
Returns $x / y$. 

**Description**

Divide two floating-point values \( x \) by \( y \) in round-to-nearest-even mode.

**Note:**

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

```c
__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 see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

```c
__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 see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
__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 finite \(x\).

**Description**

Calculate the fast approximate division of \(x\) by \(y\).

**Note:**

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

__device__ float __fmaf_ieee_rd (float x, float y, float z)
Compute fused multiply-add operation in round-down mode, ignore \(-\text{ftz}=\text{true}\) compiler flag.

**Description**

Behavior is the same as \(\_\text{fmaf}_\text{rd}(x, y, z)\), the difference is in handling denormalized inputs and outputs: \(-\text{ftz}\) 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 \(-\text{ftz}=\text{true}\) compiler flag.

**Description**

Behavior is the same as \(\_\text{fmaf}_\text{rn}(x, y, z)\), the difference is in handling denormalized inputs and outputs: \(-\text{ftz}\) 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.

- \(\text{fmaf}(\pm\infty, \pm0, z)\) returns NaN.
- \(\text{fmaf}(\pm0, \pm\infty, z)\) returns NaN.
- \(\text{fmaf}(x, y, -\infty)\) returns NaN if \(x \times y\) is an exact \(+\infty\).
- \(\text{fmaf}(x, y, +\infty)\) returns NaN if \(x \times y\) is an exact \(-\infty\).

Description
Computes the value of \(x \times y + z\) as a single ternary operation, rounding the result once in round-down (to negative infinity) mode.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
__device__ float __fmaf_rn (float x, float y, float z)
Compute \( x \times y + z \) as a single operation, in round-to-nearest-even mode.

Returns
Returns the rounded value of \( x \times y + z \) as a single operation.

- \( \text{fmaf}( \pm \infty, \pm 0, z) \) returns NaN.
- \( \text{fmaf}( \pm 0, \pm \infty, z) \) returns NaN.
- \( \text{fmaf}(x, y, -\infty) \) 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-to-nearest-even mode.

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

__device__ float __fmaf_ru (float x, float y, float z)
Compute \( x \times y + z \) 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.
__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 see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.

__device__ float __fmul_rd (float x, float y)
Multiply two floating-point values in round-down mode.

Returns
Returns \(x \times y\).

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

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.
__device__ float __fmul_rn (float x, float y)
Multiply two floating-point values in round-to-nearest-even mode.

Returns
Returns $x * y$.

Description
Compute the product of $x$ and $y$ in round-to-nearest-even mode.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.

__device__ float __fmul_ru (float x, float y)
Multiply two floating-point values in round-up mode.

Returns
Returns $x * y$.

Description
Compute the product of $x$ and $y$ in round-up (to positive infinity) mode.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.

__device__ float __fmul_rz (float x, float y)
Multiply two floating-point values in round-towards-zero mode.

Returns
Returns $x * y$. 

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.
Description
Compute the product of $x$ and $y$ in round-towards-zero mode.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.

__device__ float __frcp_rd (float x)
Compute $\frac{1}{x}$ in round-down mode.

Returns
Returns $\frac{1}{x}$.

Description
Compute the reciprocal of $x$ in round-down (to negative infinity) mode.

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

__device__ float __frcp_rn (float x)
Compute $\frac{1}{x}$ in round-to-nearest-even mode.

Returns
Returns $\frac{1}{x}$.

Description
Compute the reciprocal of $x$ in round-to-nearest-even mode.

Note:
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
__device__ float __frcp_ru (float x)

Compute $\frac{1}{x}$ in round-up mode.

Returns

Returns $\frac{1}{x}$.

Description

Compute the reciprocal of $x$ in round-up (to positive infinity) mode.

Note:

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

__device__ float __frcp_rz (float x)

Compute $\frac{1}{x}$ in round-towards-zero mode.

Returns

Returns $\frac{1}{x}$.

Description

Compute the reciprocal of $x$ in round-towards-zero mode.

Note:

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

__device__ float __frsqrt_rn (float x)

Compute $\frac{1}{\sqrt{x}}$ in round-to-nearest-even mode.

Returns

Returns $\frac{1}{\sqrt{x}}$. 
Description
Compute the reciprocal square root of $x$ in round-to-nearest-even mode.

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

```
__device__ float __fsqrt_rd (float x)
```
Compute $\sqrt{x}$ in round-down mode.

**Returns**
Returns $\sqrt{x}$.

Description
Compute the square root of $x$ in round-down (to negative infinity) mode.

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

```
__device__ float __fsqrt_rn (float x)
```
Compute $\sqrt{x}$ in round-to-nearest-even mode.

**Returns**
Returns $\sqrt{x}$.

Description
Compute the square root of $x$ in round-to-nearest-even mode.

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

Compute $\sqrt{x}$ in round-up mode.

**Returns**

Returns $\sqrt{x}$.

**Description**

Compute the square root of $x$ in round-up (to positive infinity) mode.

---

**__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.

---

**__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.
__device__ float __fsub_rn (float x, float y)
Subtract two floating-point values in round-to-nearest-even mode.

Returns
Returns \( x - y \).

Description
Compute the difference of \( x \) and \( y \) in round-to-nearest-even rounding mode.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.

__device__ float __fsub_ru (float x, float y)
Subtract two floating-point values in round-up mode.

Returns
Returns \( x - y \).

Description
Compute the difference of \( x \) and \( y \) in round-up (to positive infinity) mode.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.
\texttt{cuda\_math\_api}\n\\texttt{\_device\_float\_fsub\_rz (float x, float y)}\nSubtract two floating-point values in round-towards-zero mode.

\textbf{Returns}\nReturns \( x - y \).

\textbf{Description}\nCompute the difference of \( x \) and \( y \) in round-towards-zero mode.

\textbf{Note:}\n\begin{itemize}
  \item For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Single-Precision Floating-Point Functions section.
  \item This operation will never be merged into a single multiply-add instruction.
\end{itemize}

\texttt{cuda\_math\_api}\n\\texttt{\_device\_float\_log10f (float x)}\nCalculate the fast approximate base 10 logarithm of the input argument.

\textbf{Returns}\nReturns an approximation to \( \log_{10}(x) \).

\textbf{Description}\nCalculate the fast approximate base 10 logarithm of the input argument \( x \).

\textbf{Note:}\nFor accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Intrinsic Functions section.

\texttt{cuda\_math\_api}\n\\texttt{\_device\_float\_log2f (float x)}\nCalculate the fast approximate base 2 logarithm of the input argument.

\textbf{Returns}\nReturns an approximation to \( \log_{2}(x) \).

\textbf{Description}\nCalculate the fast approximate base 2 logarithm of the input argument \( x \).
**Note:**
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Intrinsic Functions section.

```__device__ float __logf (float x)```
Calculate the fast approximate base $e$ logarithm of the input argument.

**Returns**
Returns an approximation to $\log_e(x)$.

**Description**
Calculate the fast approximate base $e$ logarithm of the input argument $x$.

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

```__device__ float __powf (float x, float y)```
Calculate the fast approximate of $x^y$.

**Returns**
Returns an approximation to $x^y$.

**Description**
Calculate the fast approximate of $x$, the first input argument, raised to the power of $y$, the second input argument, $x^y$.

**Note:**
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Intrinsic Functions section.
__device__ float __saturatef (float x)
Clamp the input argument to [+0.0, 1.0].

Returns

- __saturatef(x) returns 0 if x < 0.
- __saturatef(x) returns 1 if x > 1.
- __saturatef(x) returns x if 0 ≤ x ≤ 1.
- __saturatef(NaN) returns 0.

Description
Clamp the input argument x to be within the interval [+0.0, 1.0].

__device__ void __sincosf (float x, float *sptr, float *cptr)
Calculate the fast approximate of sine and cosine of the first input argument.

Returns

- none

Description
Calculate the fast approximate of sine and cosine of the first input argument x (measured in radians). The results for sine and cosine are written into the second argument, sptr, and, respectively, third argument, cptr.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Intrinsic Functions section.
- Denorm input/output is flushed to sign preserving 0.0.

__device__ float __sinf (float x)
Calculate the fast approximate sine of the input argument.

Returns
Returns the approximate sine of x.
**Description**
Calculate the fast approximate sine of the input argument $x$, measured in radians.

**Note:**
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Intrinsic Functions section.
- Output in the denormal range is flushed to sign preserving 0.0.

__device__ float __tanf (float x)
Calculate the fast approximate tangent of the input argument.

**Returns**
Returns the approximate tangent of $x$.

**Description**
Calculate the fast approximate tangent of the input argument $x$, measured in radians.

**Note:**
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Intrinsic Functions section.
- The result is computed as the fast divide of __sinf[] by __cosf[]. Denormal output is flushed to sign-preserving 0.0.

### 1.8. 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 see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.

```c
__device__ double __dadd_rn (double x, double y)
```
Add two floating-point values in round-to-nearest-even mode.

Returns
Returns $x + y$.

Description
Adds two floating-point values $x$ and $y$ in round-to-nearest-even mode.

**Note:**
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.

```c
__device__ double __dadd_ru (double x, double y)
```
Add two floating-point values in round-up mode.

Returns
Returns $x + y$.

Description
Adds two floating-point values $x$ and $y$ in round-up (to positive infinity) mode.

**Note:**
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
\_\_device\_\_ double \_\_dadd\_\_rz (double x, double y)
Add two floating-point values in round-towards-zero mode.

Returns
Returns x + y.

Description
Adds two floating-point values x and y in round-towards-zero mode.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.

\_\_device\_\_ double \_\_ddiv\_\_rd (double x, double y)
Divide two floating-point values in round-down mode.

Returns
Returns x / y.

Description
Divides two floating-point values x by y in round-down (to negative infinity) mode.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- Requires compute capability >= 2.0.

\_\_device\_\_ double \_\_ddiv\_\_rn (double x, double y)
Divide two floating-point values in round-to-nearest-even mode.

Returns
Returns x / y.
Description
Divides two floating-point values $x$ by $y$ in round-to-nearest-even mode.

**Note:**
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- Requires compute capability $\geq 2.0$.

```c
__device__ double __ddiv_ru (double x, double y)
```
Divide two floating-point values in round-up mode.

**Returns**
Returns $x / y$.

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

**Note:**
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- Requires compute capability $\geq 2.0$.

```c
__device__ double __ddiv_rz (double x, double y)
```
Divide two floating-point values in round-towards-zero mode.

**Returns**
Returns $x / y$.

**Description**
Divides two floating-point values $x$ by $y$ in round-towards-zero mode.

**Note:**
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
__device__ double __dmul_rd (double x, double y)
Multiply two floating-point values in round-down mode.

Returns
Returns x * y.

Description
Multiplies two floating-point values x and y in round-down (to negative infinity) mode.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.

__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 see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.

__device__ double __dmul_ru (double x, double y)
Multiply two floating-point values in round-up mode.

Returns
Returns x * y.
Description
Multiplies two floating-point values \( x \) and \( y \) in round-up (to positive infinity) mode.

**Note:**
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.

```c
__device__ double __dmul_rz (double x, double y)
```
Multiply two floating-point values in round-towards-zero mode.

**Returns**
Returns \( x \times y \).

**Description**
Multiplies two floating-point values \( x \) and \( y \) in round-towards-zero mode.

**Note:**
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.

```c
__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.
__device__ double __drcp_rn (double x)
Compute $\frac{1}{x}$ in round-to-nearest-even mode.

Returns
Returns $\frac{1}{x}$.

Description
Compute the reciprocal of $x$ in round-to-nearest-even mode.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- Requires compute capability $\geq$ 2.0.

__device__ double __drcp_ru (double x)
Compute $\frac{1}{x}$ in round-up mode.

Returns
Returns $\frac{1}{x}$.

Description
Compute the reciprocal of $x$ in round-up (to positive infinity) mode.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- Requires compute capability $\geq$ 2.0.
__device__ double __drcp_rz (double x)
Compute $\frac{1}{x}$ in round-towards-zero mode.

Returns
Returns $\frac{1}{x}$.

Description
Compute the reciprocal of $x$ in round-towards-zero mode.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- Requires compute capability >= 2.0.

__device__ double __dsqrt_rd (double x)
Compute $\sqrt{x}$ in round-down mode.

Returns
Returns $\sqrt{x}$.

Description
Compute the square root of $x$ in round-down (to negative infinity) mode.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- Requires compute capability >= 2.0.

__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 see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- Requires compute capability >= 2.0.

```
__device__ double __dsqrt_ru (double x)
```

Description
Compute the square root of x in round-up mode.

**Returns**
Returns $\sqrt{x}$.

```
__device__ double __dsqrt_rz (double x)
```

Description
Compute the square root of x in round-towards-zero mode.

**Note:**
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- Requires compute capability >= 2.0.
__device__ double __dsub_rd (double x, double y)
Subtract two floating-point values in round-down mode.

Returns
Returns \( x - y \).

Description
Subtracts two floating-point values \( x \) and \( y \) in round-down (to negative infinity) mode.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.

__device__ double __dsub_rn (double x, double y)
Subtract two floating-point values in round-to-nearest-even mode.

Returns
Returns \( x - y \).

Description
Subtracts two floating-point values \( x \) and \( y \) in round-to-nearest-even mode.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.
__device__ double __dsub_ru (double x, double y)
Subtract two floating-point values in round-up mode.

Returns
Returns x - y.

Description
Subtracts two floating-point values x and y in round-up (to positive infinity) mode.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.

__device__ double __dsub_rz (double x, double y)
Subtract two floating-point values in round-towards-zero mode.

Returns
Returns x - y.

Description
Subtracts two floating-point values x and y in round-towards-zero mode.

Note:
- For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
- This operation will never be merged into a single multiply-add instruction.

__device__ double __fma_rd (double x, double y, double z)
Compute \( x \times y + z \) as a single operation in round-down mode.

Returns
Returns the rounded value of \( x \times y + z \) as a single operation.
- `fmaf(\pm\infty, \pm0, z)` returns NaN.
- `fmaf(\pm0, \pm\infty, z)` returns NaN.
- `fmaf(x, y, -\infty)` returns NaN if \(xy\) is an exact \(+\infty\)
- `fmaf(x, y, +\infty)` returns NaN if \(xy\) is an exact \(-\infty\)

**Description**
Computes the value of \(xy + z\) as a single ternary operation, rounding the result once in round-down (to negative infinity) mode.

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

```
__device__ double __fma_rn (double x, double y, double z)
```
Compute \(xy + z\) as a single operation in round-to-nearest-even mode.

**Returns**
Returns the rounded value of \(xy + z\) as a single operation.
- `fmaf(\pm\infty, \pm0, z)` returns NaN.
- `fmaf(\pm0, \pm\infty, z)` returns NaN.
- `fmaf(x, y, -\infty)` returns NaN if \(xy\) is an exact \(+\infty\)
- `fmaf(x, y, +\infty)` returns NaN if \(xy\) is an exact \(-\infty\)

**Description**
Computes the value of \(xy + z\) as a single ternary operation, rounding the result once in round-to-nearest-even mode.

**Note:**
For accuracy information see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.
__device__ double __fma_ru (double x, double y, double z)

Compute \( x \times y + z \) as a single operation in round-up mode.

**Returns**

Returns the rounded value of \( x \times y + z \) as a single operation.

- \( \text{fmaf} (\pm \infty, \pm 0, z) \) 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 see the CUDA C++ Programming Guide, Mathematical Functions Appendix, Double-Precision Floating-Point Functions section.

__device__ double __fma_rz (double x, double y, double z)

Compute \( x \times y + z \) 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.
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: \( \text{result}[n] := \text{input}[\text{selector}[n]] \) where \( \text{result}[n] \) is the \( n \)th 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.

\[
\text{__device__ int __ffsll (long long int } x)\]
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.

- \( \text{__ffsll(0)} \) 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.

\[
\text{__device__ unsigned int __funnelshift_l (unsigned int } lo, \text{ unsigned int } hi, \text{ unsigned int } shift)\]
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.

\[
\text{__device__ unsigned int __funnelshift_lc (unsigned int } lo, \text{ unsigned int } hi, \text{ unsigned int } shift)\]
Concatenate \( hi:lo \), shift left by \( \text{min(shift, 32)} \) 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` (`shift` & 31). The least significant 32-bits of the result are returned.

```
__device__ unsigned int __funnelshift_rc (unsigned int lo, unsigned int hi, unsigned int shift)
```

Concatenate `hi : lo`, shift right by `min(`shift`, 32)` bits, return the least significant 32 bits.

**Returns**
Returns the least significant 32 bits of the shifted 64-bit value.

**Description**
Shift the 64-bit value formed by concatenating argument `lo` and `hi` right by the amount specified by the argument `shift`. Argument `lo` holds bits 31:0 and argument `hi` holds bits 63:32 of the 64-bit source value. The source is shifted right by the clamped value of `shift` (min(`shift`, 32)). The least significant 32-bits of the result are returned.

```
__device__ int __hadd (int x, int y)
```

Compute average of signed input arguments, avoiding overflow in the intermediate sum.

**Returns**
Returns a signed integer value representing the signed average value of the two inputs.
Description
Compute average of signed input arguments x and y as \((x + y) \gg 1\), avoiding overflow in the intermediate sum.

__device__ int __mul24 (int x, int y)
Calculate the least significant 32 bits of the product of the least significant 24 bits of two integers.

Returns
Returns the least significant 32 bits of the product \(x \times y\).

Description
Calculate the least significant 32 bits of the product of the least significant 24 bits of \(x\) and \(y\). The high order 8 bits of \(x\) and \(y\) are ignored.

__device__ long long int __mul64hi (long long int x, long long int y)
Calculate the most significant 64 bits of the product of the two 64-bit integers.

Returns
Returns the most significant 64 bits of the product \(x \times y\).

Description
Calculate the most significant 64 bits of the 128-bit product \(x \times y\), where \(x\) and \(y\) are 64-bit integers.

__device__ int __mulhi (int x, int y)
Calculate the most significant 32 bits of the product of the two 32-bit integers.

Returns
Returns the most significant 32 bits of the product \(x \times y\).

Description
Calculate the most significant 32 bits of the 64-bit product \(x \times y\), where \(x\) and \(y\) are 32-bit integers.
__device__ int __popc (unsigned int x)
Count the number of bits that are set to 1 in a 32-bit integer.

Returns
Returns a value between 0 and 32 inclusive representing the number of set bits.

Description
Count the number of bits that are set to 1 in \( x \).

__device__ int __popcll (unsigned long long int x)
Count the number of bits that are set to 1 in a 64-bit integer.

Returns
Returns a value between 0 and 64 inclusive representing the number of set bits.

Description
Count the number of bits that are set to 1 in \( x \).

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

Returns
Returns a signed integer value representing the signed rounded average value of the two inputs.

Description
Compute average of signed input arguments \( x \) and \( y \) as \( \lfloor \frac{x + y + 1}{2} \rfloor \), avoiding overflow in the intermediate sum.

__device__ unsigned int __sad (int x, int y, unsigned int z)
Calculate \( |x - y| + z \), the sum of absolute difference.

Returns
Returns \( |x - y| + z \).
Description
Calculate |x − y| + z, the 32-bit sum of the third argument z plus and the absolute value of the difference between the first argument, x, and second argument, y.
Inputs x and y are signed 32-bit integers, input z is a 32-bit unsigned integer.

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

Returns
Returns an unsigned integer value representing the unsigned average value of the two inputs.

Description
Compute average of unsigned input arguments x and y as (x + y) >> 1, avoiding overflow in the intermediate sum.

__device__ unsigned int __umul24 (unsigned int x, unsigned int y)
Calculate the least significant 32 bits of the product of the least significant 24 bits of two unsigned integers.

Returns
Returns the least significant 32 bits of the product x * y.

Description
Calculate the least significant 32 bits of the product of the least significant 24 bits of two unsigned integers. The high order 8 bits of x and y are ignored.

__device__ unsigned long long int __umul64hi (unsigned long long int x, unsigned long long int y)
Calculate the most significant 64 bits of the product of the two 64 unsigned bit integers.

Returns
Returns the most significant 64 bits of the product x * y.
Description
Calculate the most significant 64 bits of the 128-bit product $x \times y$, where $x$ and $y$ are 64-bit unsigned integers.

```
__device__ unsigned int __umulhi (unsigned int x, unsigned int y)
```

Calculate the most significant 64 bits of the 128-bit product $x \times y$, where $x$ and $y$ are 64-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__ unsigned int __urhadd (unsigned int x, unsigned int y)
```

Compute rounded average of unsigned input arguments, avoiding overflow in the intermediate sum.

Returns
Returns an unsigned integer value representing the unsigned rounded average value of the two inputs.

Description
Compute average of unsigned input arguments $x$ and $y$ as $(x + y + 1) >> 1$, avoiding overflow in the intermediate sum.

```
__device__ unsigned int __usad (unsigned int x, unsigned 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 $x$, $y$, and $z$ are unsigned 32-bit integers.

1.10. Type Casting Intrinsics

This section describes type casting intrinsic functions that are only supported in device code. To use these functions you do not need to include any additional header files in your program.

__$\text{device}\_\text{__ float } \text{__double2float\_rd} (\text{double x})$

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.

__$\text{device}\_\text{__ float } \text{__double2float\_rn} (\text{double x})$

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.

__$\text{device}\_\text{__ float } \text{__double2float\_ru} (\text{double x})$

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.

```c
__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.

```c
__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.

```c
__device__ int __double2int_rd (double x)
```
Convert a double to a signed int in round-down mode.

Returns
Returns converted value.

Description
Convert the double-precision floating-point value \( x \) to a signed integer value in round-down (to negative infinity) mode.
__device__ int __double2int_rn (double x)
Convert a double to a signed int in round-to-nearest-even mode.

Returns
Returns converted value.

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

__device__ int __double2int_ru (double x)
Convert a double to a signed int in round-up mode.

Returns
Returns converted value.

Description
Convert the double-precision floating-point value \( x \) to a signed integer value in round-up (to positive infinity) mode.

__device__ int __double2int_rz (double x)
Convert a double to a signed int in round-towards-zero mode.

Returns
Returns converted value.

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

__device__ long long int __double2ll_rd (double x)
Convert a double to a signed 64-bit int in round-down mode.

Returns
Returns converted value.
**Description**
Convert the double-precision floating-point value $x$ to a signed 64-bit integer value in round-down (to negative infinity) mode.

```c
__device__ long long int __double2ll_rn (double x)
```
Convert a double to a signed 64-bit int in round-to-nearest-even mode.

**Returns**
Returns converted value.

**Description**
Convert the double-precision floating-point value $x$ to a signed 64-bit integer value in round-to-nearest-even mode.

```c
__device__ long long int __double2ll_ru (double x)
```
Convert a double to a signed 64-bit int in round-up mode.

**Returns**
Returns converted value.

**Description**
Convert the double-precision floating-point value $x$ to a signed 64-bit integer value in round-up (to positive infinity) mode.

```c
__device__ long long int __double2ll_rz (double x)
```
Convert a double to a signed 64-bit int in round-towards-zero mode.

**Returns**
Returns converted value.

**Description**
Convert the double-precision floating-point value $x$ to a signed 64-bit integer value in round-towards-zero mode.
__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.

\[
\text{__device__ unsigned int __double2uint_rz (double x)}
\]
Convert a double to an unsigned int in round-towards-zero mode.

Returns
Returns converted value.

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

\[
\text{__device__ unsigned long long int __double2ull_rd (double x)}
\]
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.

\[
\text{__device__ unsigned long long int __double2ull_rn (double x)}
\]
Convert a double to an unsigned 64-bit int in round-to-nearest-even mode.

Returns
Returns converted value.

Description
Convert the double-precision floating-point value \( x \) to an unsigned 64-bit integer value in round-to-nearest-even mode.
__device__ unsigned long long int __double2ull_ru (double x)
Convert a double to an unsigned 64-bit int in round-up mode.

Returns
Returns converted value.

Description
Convert the double-precision floating-point value \( x \) to an unsigned 64-bit integer value in round-up (to positive infinity) mode.

__device__ unsigned long long int __double2ull_rz (double x)
Convert a double to an unsigned 64-bit int in round-towards-zero mode.

Returns
Returns converted value.

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

__device__ long long int __double_as_longlong (double x)
Reinterpret bits in a double as a 64-bit signed integer.

Returns
Returns reinterpreted value.

Description
Reinterpret the bits in the double-precision floating-point value \( x \) as a signed 64-bit integer.
__device__ int __float2int_rd (float x)
Convert a float to a signed integer in round-down mode.

Returns
Returns converted value.

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

__device__ int __float2int_rn (float x)
Convert a float to a signed integer in round-to-nearest-even mode.

Returns
Returns converted value.

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

__device__ int __float2int_ru (float)
Convert a float to a signed integer in round-up mode.

Returns
Returns converted value.

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

__device__ int __float2int_rz (float x)
Convert a float to a signed integer in round-towards-zero mode.

Returns
Returns converted value.
Description
Convert the single-precision floating-point value \( x \) to a signed integer in round-towards-zero mode.

\[
\_\_\text{device}\_\_\long\long\int\_\_\text{float2ll\_rd}\ (\float x)
\]
Convert a float to a signed 64-bit integer in round-down mode.

Returns
Returns converted value.

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

\[
\_\_\text{device}\_\_\long\long\int\_\_\text{float2ll\_rn}\ (\float x)
\]
Convert a float to a signed 64-bit integer in round-to-nearest-even mode.

Returns
Returns converted value.

Description
Convert the single-precision floating-point value \( x \) to a signed 64-bit integer in round-to-nearest-even mode.

\[
\_\_\text{device}\_\_\long\long\int\_\_\text{float2ll\_ru}\ (\float x)
\]
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 a unsigned integer.

**Returns**

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

\[
\text{__device__ double __hiloint2double (int hi, int lo)}
\]
Reinterpret high and low 32-bit integer values as a double.

Returns
Returns reinterpreted value.

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

\[
\text{__device__ double __int2double_rn (int x)}
\]
Convert a signed int to a double.

Returns
Returns converted value.

Description
Convert the signed integer value \( x \) to a double-precision floating-point value.

\[
\text{__device__ float __int2float_rd (int x)}
\]
Convert a signed integer to a float in round-down mode.

Returns
Returns converted value.

Description
Convert the signed integer value \( x \) to a single-precision floating-point value in round-down [to negative infinity] mode.
\_\text{device\_} \text{float} \_\text{int2float\_rn} \ (\text{int} \ x) \\
Convert a signed integer to a float in round-to-nearest-even mode.

\textbf{Returns}
Returns converted value.

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

\_\text{device\_} \text{float} \_\text{int2float\_ru} \ (\text{int} \ x) \\
Convert a signed integer to a float in round-up mode.

\textbf{Returns}
Returns converted value.

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

\_\text{device\_} \text{float} \_\text{int2float\_rz} \ (\text{int} \ x) \\
Convert a signed integer to a float in round-towards-zero mode.

\textbf{Returns}
Returns converted value.

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

\_\text{device\_} \text{float} \_\text{int\_as\_float} \ (\text{int} \ x) \\
Reinterpret bits in an integer as a float.

\textbf{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.

```c
__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.

```c
__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.

```c
__device__ double __uint2double_rn (unsigned int x)
```
Convert an unsigned int to a double.

Returns
Returns converted value.

Description
Convert the unsigned integer value \( x \) to a double-precision floating-point value.
__device__ float __uint2float_rd (unsigned int x)
Convert an unsigned integer to a float in round-down mode.

Returns
Returns converted value.

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

__device__ float __uint2float_rn (unsigned int x)
Convert an unsigned integer to a float in round-to-nearest-even mode.

Returns
Returns converted value.

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

__device__ float __uint2float_ru (unsigned int x)
Convert an unsigned integer to a float in round-up mode.

Returns
Returns converted value.

Description
Convert the unsigned integer value \( x \) to a single-precision floating-point value in round-up (to positive infinity) mode.

__device__ float __uint2float_rz (unsigned int x)
Convert an unsigned integer to a float in round-towards-zero mode.

Returns
Returns converted value.
Description
Convert the unsigned integer value \( x \) to a single-precision floating-point value in round-towards-zero mode.

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

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.
__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. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vabs4 (unsigned int a)
Computes per-byte absolute value.

Returns
Returns computed value.

Description
Splits argument by bytes. Computes absolute value of each byte. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vabsdiffs2 (unsigned int a, unsigned int b)
Computes per-halfword sum of absolute difference of signed integer.

Returns
Returns computed value.

Description
Splits 4 bytes of each into 2 parts, each consisting of 2 bytes. For corresponding parts function computes absolute difference. Partial results are recombined and returned as unsigned int.
__device__ unsigned int __vabsdiffs4 (unsigned int a, unsigned int b)
Computes per-byte absolute difference of signed integer.

Returns
Returns computed value.

Description
Splits 4 bytes of each into 4 parts, each consisting of 1 byte. For corresponding parts function computes absolute difference. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vabsdiffu2 (unsigned int a, unsigned int b)
Performs per-halfword absolute difference of unsigned integer computation: |a - b|.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function computes absolute difference. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vabsdiffu4 (unsigned int a, unsigned int b)
Computes per-byte absolute difference of unsigned integer.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function computes absolute difference. Partial results are recombined and returned as unsigned int.
__device__ unsigned int __vabsss2 (unsigned int a)
Computes per-halfword absolute value with signed saturation.

Returns
Returns computed value.

Description
Splits 4 bytes of argument into 2 parts, each consisting of 2 bytes, then computes absolute value with signed saturation for each of parts. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vabsss4 (unsigned int a)
Computes per-byte absolute value with signed saturation.

Returns
Returns computed value.

Description
Splits 4 bytes of argument into 4 parts, each consisting of 1 byte, then computes absolute value with signed saturation for each of parts. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vadd2 (unsigned int a, unsigned int b)
Performs per-halfword (un)signed addition, with wrap-around: a + b.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes, then performs unsigned addition on corresponding parts. Partial results are recombined and returned as unsigned int.
__device__ unsigned int __vadd4 (unsigned int a, unsigned int b)
Performs per-byte (un)signed addition.

**Returns**
Returns computed value.

**Description**
Splits 'a' into 4 bytes, then performs unsigned addition on each of these bytes with the corresponding byte from 'b', ignoring overflow. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vaddss2 (unsigned int a, unsigned int b)
Performs per-halfword addition with signed saturation.

**Returns**
Returns computed value.

**Description**
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes, then performs addition with signed saturation on corresponding parts. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vaddss4 (unsigned int a, unsigned int b)
Performs per-byte addition with signed saturation.

**Returns**
Returns computed value.

**Description**
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte, then performs addition with signed saturation on corresponding parts. Partial results are recombined and returned as unsigned int.
__device__ unsigned int __vaddus2 (unsigned int a, unsigned int b)
Performs per-halfword addition with unsigned saturation.

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.

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.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes, then computes signed rounded average of corresponding parts. Partial results are recombined and returned as unsigned int.
__device__ unsigned int __vavgs4 (unsigned int a, unsigned int b)
Computes per-byte signed rounded average.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. Then computes signed rounded average of corresponding parts. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vavgu2 (unsigned int a, unsigned int b)
Performs per-halfword unsigned rounded average computation.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes, then computes unsigned rounded average of corresponding parts. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vavgu4 (unsigned int a, unsigned int b)
Performs per-byte unsigned rounded average.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte, then computes unsigned rounded average of corresponding parts. Partial results are recombined and returned as unsigned int.
__device__ unsigned int __vcmpeq2 (unsigned int a, unsigned int b)
Performs per-halfword (un)signed comparison.

Returns
Returns 0xffff computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts result is ffff if they are equal, and 0000 otherwise. For example __vcmpeq2(0x1234aba5, 0x1234aba6) returns 0xffff0000.

__device__ unsigned int __vcmpeq4 (unsigned int a, unsigned int b)
Performs per-byte (un)signed comparison.

Returns
Returns 0xff if a = b, else returns 0.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts result is ff if they are equal, and 00 otherwise. For example __vcmpeq4(0x1234aba5, 0x1234aba6) returns 0xffffff00.

__device__ unsigned int __vcmpges2 (unsigned int a, unsigned int b)
Performs per-halfword signed comparison: a >= b ? 0xffff : 0.

Returns
Returns 0xffff if a >= b, else returns 0.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts result is ffff if 'a' part >= 'b' part, and 0000 otherwise. For example __vcmpges2(0x1234aba5, 0x1234aba6) returns 0xffff0000.
__device__ unsigned int __vcmpges4 (unsigned int a, unsigned int b)
Performs per-byte signed comparison.

Returns
Returns 0xff if a >= b, else returns 0.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts result is ff if ‘a’ part >= ‘b’ part, and 00 otherwise. For example __vcmpges4(0x1234aba5, 0x1234aba6) returns 0xffffffff.

__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 0xffffffff.

__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 0xffffffff.
__device__ unsigned int __vcmpgts2 (unsigned int a, unsigned int b)
Performs per-halfword signed comparison: $a > b \rightarrow 0xffff : 0$.

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: $a > b \rightarrow 0xffff : 0$.

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 0xffffff if a <= b, else returns 0.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts result is ffff if 'a' part <= 'b' part, and 0000 otherwise. For example __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: \( a \leq b \) ? 0xffff : 0.

Returns
Returns 0xffff if \( a \leq b \), 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 \( \leq \) ‘b’ part, and 0000 otherwise. For example __vcmpleu2(0x1234aba5, 0x1234aba6) returns 0xffffffff.

__device__ unsigned int __vcmpleu4 (unsigned int a, unsigned int b)
Performs per-byte unsigned comparison.

Returns
Returns 0xff if \( a \leq b \), else returns 0.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts result is ff if ‘a’ part \( \leq \) ‘b’ part, and 00 otherwise. For example __vcmpleu4(0x1234aba5, 0x1234aba6) returns 0xffffffff.

__device__ unsigned int __vcmplts2 (unsigned int a, unsigned int b)
Performs per-halfword signed comparison: \( a < b \) ? 0xffff : 0.

Returns
Returns 0xffff if \( a < b \), else returns 0.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts result is ffff if ‘a’ part \( < \) ‘b’ part, and 0000 otherwise. For example __vcmplts2(0x1234aba5, 0x1234aba6) returns 0x0000ffff.
__device__ unsigned int __vcmplts4 (unsigned int a, unsigned int b)
Performs per-byte signed comparison.

Returns
Returns 0xff if \(a < b\), else returns 0.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts result is ff if \('a'\) part < \('b'\) part, and 00 otherwise. For example __vcmplts4(0x1234aba5, 0x1234aba6) returns 0x000000ff.

__device__ unsigned int __vcmpltu2 (unsigned int a, unsigned int b)
Performs per-halfword unsigned comparison: \(a < b\) ? 0xffff : 0.

Returns
Returns 0xffff if \(a < b\), else returns 0.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts result is ffff if \('a'\) part < \('b'\) part, and 0000 otherwise. For example __vcmpltu2(0x1234aba5, 0x1234aba6) returns 0x0000ffff.

__device__ unsigned int __vcmpltu4 (unsigned int a, unsigned int b)
Performs per-byte unsigned comparison.

Returns
Returns 0xff if \(a < b\), else returns 0.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts result is ff if \('a'\) part < \('b'\) part, and 00 otherwise. For example __vcmpltu4(0x1234aba5, 0x1234aba6) returns 0x000000ff.
__device__ unsigned int __vcmpne2 (unsigned int a, unsigned int b)
Performs per-halfword (un)signed comparison: a != b ? 0xffff : 0.

Returns
Returns 0xffff if a != b, else returns 0.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts result is ffff if ‘a’ part != ‘b’ part, and 0000 otherwise. For example __vcmplts2(0x1234aba5, 0x1234aba6) returns 0x0000ffff.

__device__ unsigned int __vcmpne4 (unsigned int a, unsigned int b)
Performs per-byte (un)signed comparison.

Returns
Returns 0xff if a != b, else returns 0.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts result is ff if ‘a’ part != ‘b’ part, and 00 otherwise. For example __vcmplts4(0x1234aba5, 0x1234aba6) returns 0x000000ff.

__device__ unsigned int __vhaddu2 (unsigned int a, unsigned int b)
Performs per-halfword unsigned average computation.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes, then computes unsigned average of corresponding parts. Partial results are recombined and returned as unsigned int.
__device__ unsigned int __vhaddu4 (unsigned int a, unsigned int b)
Computes per-byte unsigned average.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. then computes unsigned average of corresponding parts. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vmaxs2 (unsigned int a, unsigned int b)
Performs per-halfword signed maximum computation.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function computes signed maximum. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vmaxs4 (unsigned int a, unsigned int b)
Computes per-byte signed maximum.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function computes signed maximum. Partial results are recombined and returned as unsigned int.
__device__ unsigned int __vmaxu2 (unsigned int a, unsigned int b)
Performs per-halfword unsigned maximum computation.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function computes unsigned maximum. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vmaxu4 (unsigned int a, unsigned int b)
Computes per-byte unsigned maximum.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function computes unsigned maximum. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vmins2 (unsigned int a, unsigned int b)
Performs per-halfword signed minimum computation.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function computes signed minimum. Partial results are recombined and returned as unsigned int.
__device__ unsigned int __vmins4 (unsigned int a, unsigned int b)
Computes per-byte signed minimum.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function computes signed minimum. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vminu2 (unsigned int a, unsigned int b)
Performs per-halfword unsigned minimum computation.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function computes unsigned minimum. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vminu4 (unsigned int a, unsigned int b)
Computes per-byte unsigned minimum.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function computes unsigned minimum. Partial results are recombined and returned as unsigned int.
__device__ unsigned int __vneg2 (unsigned int a)
Computes per-halfword negation.

Returns
Returns computed value.

Description
Splits 4 bytes of argument into 2 parts, each consisting of 2 bytes. For each part function computes negation. Partial results are recombed and returned as unsigned int.

__device__ unsigned int __vneg4 (unsigned int a)
Performs per-byte negation.

Returns
Returns computed value.

Description
Splits 4 bytes of argument into 4 parts, each consisting of 1 byte. For each part function computes negation. Partial results are recombed and returned as unsigned int.

__device__ unsigned int __vnegss2 (unsigned int a)
Computes per-halfword negation with signed saturation.

Returns
Returns computed value.

Description
Splits 4 bytes of argument into 2 parts, each consisting of 2 bytes. For each part function computes negation. Partial results are recombed and returned as unsigned int.

__device__ unsigned int __vnegss4 (unsigned int a)
Performs per-byte negation with signed saturation.

Returns
Returns computed value.
Description
Splits 4 bytes of argument into 4 parts, each consisting of 1 byte. For each part function computes negation. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vsads2 (unsigned int a, unsigned int b)
Performs per-halfword sum of absolute difference of signed.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function computes absolute difference and sum it up. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vsads4 (unsigned int a, unsigned int b)
Computes per-byte sum of abs difference of signed.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function computes absolute difference and sum it up. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vsadu2 (unsigned int a, unsigned int b)
Computes per-halfword sum of abs diff of unsigned.

Returns
Returns computed value.
**Description**

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function computes absolute differences and returns sum of those differences.

```c
__device__ unsigned int __vsadu4 (unsigned int a, unsigned int b)
```

Computes per-byte sum of abs difference of unsigned.

**Returns**

Returns computed value.

**Description**

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function computes absolute differences and returns sum of those differences.

```c
__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.

```c
__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.

_dev__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.

_dev__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.

_dev__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.
__device__ unsigned int __vsetlt2 (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 __vsetlt4 (unsigned int a, unsigned int b)
Performs per-byte signed comparison.

Returns
Returns 1 if a < b, else returns 0.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function performs comparison ‘a’ part <= ‘b’ part. If both inequalities are satisfied, function returns 1.

__device__ unsigned int __vsetltu2 (unsigned int a, unsigned int b)
Performs per-halfword unsigned comparison.

Returns
Returns 1 if a < b, else returns 0.
Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function performs comparison ‘a’ part <= ‘b’ part. If both inequalities are satisfied, function returns 1.

__device__ unsigned int __vsetltu4 (unsigned int a, unsigned int b)
Performs per-byte unsigned comparison.

Returns
Returns 1 if a < b, else returns 0.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function performs comparison ‘a’ part <= ‘b’ part. If both inequalities are satisfied, function returns 1.

__device__ unsigned int __vsetne2 (unsigned int a, unsigned int b)
Performs per-halfword (un)signed comparison.

Returns
Returns 1 if a != b, else returns 0.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function performs comparison ‘a’ part != ‘b’ part. If both conditions are satisfied, function returns 1.

__device__ unsigned int __vsetne4 (unsigned int a, unsigned int b)
Performs per-byte (un)signed comparison.

Returns
Returns 1 if a != b, else returns 0.
Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function performs comparison 'a' part != 'b' part. If both conditions are satisfied, function returns 1.

__device__ unsigned int __vsub2 (unsigned int a, unsigned int b)
Performs per-halfword (un)signed subtraction, with wrap-around.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function performs subtraction. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vsub4 (unsigned int a, unsigned int b)
Performs per-byte subtraction.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function performs subtraction. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vsubss2 (unsigned int a, unsigned int b)
Performs per-halfword (un)signed subtraction, with signed saturation.

Returns
Returns computed value.
Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function performs subtraction with signed saturation. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vsubss4 (unsigned int a, unsigned int b)
Performs per-byte subtraction with signed saturation.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function performs subtraction with signed saturation. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vsubus2 (unsigned int a, unsigned int b)
Performs per-halfword subtraction with unsigned saturation.

Returns
Returns computed value.

Description
Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function performs subtraction with unsigned saturation. Partial results are recombined and returned as unsigned int.

__device__ unsigned int __vsubus4 (unsigned int a, unsigned int b)
Performs per-byte subtraction with unsigned saturation.

Returns
Returns computed value.
Description
Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function performs subtraction with unsigned saturation. Partial results are recombined and returned as unsigned int.
Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation ("NVIDIA") makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

OpenCL

OpenCL is a trademark of Apple Inc. used under license to the Khronos Group Inc.

Trademarks

NVIDIA and the NVIDIA logo are trademarks or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

© 2007-2022 NVIDIA Corporation & affiliates. All rights reserved.