

cudaMemcpy2D(dst, dpitch, src, spitch, width, height, kdir)
cudaMemcpy2DArrayToArray(dsta, dstx, dsty, srca, srcx, srcy, width, height, kdir)
cudaMemcpy2DAsync(dst, dpitch, src, spitch, width, height, kdir, stream)
cudaMemcpy2DFromArray(dst, dpitch, srca, srcx, srcy, width, height, kdir)
cudaMemcpy2DToArray(dsta, dstx, dsty, src, spitch, width, height, kdir)
cudaMemcpy3D(p)
cudaMemcpy3DAsync(p, stream)
cudaMemcpy3DPeer(p)
cudaMemcpy3DPeerAsync(p, stream)
cudaMemGetInfo(free, total)
cudaMemset(devptr, value, count)
cudaMemset2D(devptr, pitch, value, width, height)
cudaMemset3D(pitchptr, value, cext)

API Routines for Execution Control

cudaFuncGetAttributes(attr, func)
cudaFuncSetCacheConfig(func, cacheconfig)
cudaSetDoubleForDevice(d)
cudaSetDoubleForHost(d)

API Routines for Stream Management

cudaStreamCreate(stream)
cudaStreamDestroy(stream)
cudaStreamQuery(stream)
cudaStreamSynchronize(stream)
cudaStreamWaitEvent(stream, events, flags)

API Routines for Event Management

cudaEventCreate(event)
cudaEventCreateWithFlags(event, flags)
cudaEventDestroy(event)
cudaEventElapsedTime(time, start, end)
cudaEventQuery(event)
cudaEventRecord(event, stream)
cudaEventSynchronize(event)

API Routines for Unified Addressing and Peer Device Memory Access

cudaDeviceCanAccessPeer(canAccessPeer, device, peerDevice)
cudaDeviceDisablePeerAccess(peerDevice)
cudaDeviceEnablePeerAccess(peerDevice, flags)
cudaPointerGetAttributes(attr, ptr)

API Routines for Error Handling

cudaGetErrorString(errorcode)
cudaGetLastError()
cudaPeekAtLastError()

API Routines for Version Management

cudaDriverGetVersion(version)
cudaRuntimeGetVersion(version)

Compiler Options

pgfortran a.cuf -Mcuda[=options]

Where options are one or more of the following seprated by commas:

- cc13|cc1x|cc20|cc2x
- cuda3.0|cuda3.1|cuda3.2|cuda4.0
- fastmath
- maxregcount:n
- nofma
- ptxinfo

Example

```
module curve_mod
  real, constant :: twopi
  contains
    attributes(global) subroutine curve(x, n)
      real, device :: x(n)
      integer, value :: n
      real, shared :: term
      itx = threadidx%x
      if (itx == 0) term = twopi / float(n)
      call syncthreads()
      i = (blockidx%x-1)*blockdim%x + itx
      if (i <= n) then
        x(i) = cos(float(i-1)*term)
      end if
      return
    end subroutine
end module
```

```
program qrc_example
  use cublas
  use curve_mod
  integer, parameter :: N = 20000
  real, device :: x(N)
  twopi = atan(1.0)*8
  call curve<<<(N-1)/512+1,512>>>(x, N)
  !$cuf kernel do <<< *, * >>>
  do i = 1, N
    x(i) = 2.0 * x(i) * x(i)
  end do
  print *, "float(N) = ", sasum(N,x,1)
end
```



www.pgroup.com/cudafortran

© 2011 The Portland Group, Inc. All rights reserved.

CUDA Fortran

Quick Reference Card

CUDA Fortran is a Fortran analog to the NVIDIA CUDA C

language for programming GPUs. It includes language

features, intrinsic functions and API routines for

writing CUDA kernels and host control code in Fortran

while remaining fully interoperable with CUDA C. Included

in the language are subroutine attributes to define global

(kernel) and device subroutines and functions, and

variable attributes to declare and allocate device data and

host pinned data. Most Fortran numeric and mathemati-

cal intrinsic functions can be used in device code, as can

many CUDA device built-in and libm functions. The CUDA

Fortran language allows allocatable device arrays, and

array assignments between host and device arrays using

standard Fortran syntax to move data between host and

device memory. A full set of CUDA runtime API routines is

available for low-level control of device memory, streams,

asynchronous operations, and events.

Subprogram Qualifiers

attributes(global)

- declares a subroutine as being a kernel
- must be a SUBROUTINE
- may not be recursive
- may not contain variables with SAVE attribute or data initialization
- may not have OPTIONAL arguments

attributes(device)

- declares a subprogram that is executed on and callable only from the device
- may not be recursive
- may not contain variables with the SAVE attribute or data initialization
- may not have OPTIONAL arguments

Variable Qualifiers

device

- allocated in device global memory

constant

- allocated in device constant memory space
- may be written by a host subprogram; is read-only in global and device subprograms

shared

- allocated in device shared memory
- may only appear in a global or device subprogram

pinned

- allocated in host page-locked memory
- must be an allocatable array; valid only in host subprograms

Predefined Types, Constants and Variables

(Parenthesized names used in Host Control API descriptions)

Parameters

cuda_count_kind (count)
cuda_stream_kind (stream)

Types

C_DEVPTR (devptr)	cudaArrayPtr (carray)
DIM3 (dim3)	cudaChannelFormatDesc (cdesc)
cudaDeviceProp (prop)	cudaFuncAttributes (attr)
cudaEvent (event)	cudaMemcpy3DParms (p)
cudaExtent (cext)	cudaMemcpy3DPeerParms (p)
cudaPos	cudaPitchedPtr (pitchptr)
cudaSymbol (symbol)	cudaPointerAttributes (ptr)

Variables

type(dim3) :: threadidx, blockdim, blockidx, griddim
integer(kind=4) :: warpsize

Device Code

Data Types

character(len= 1)	integer(kind= 1 2 4 8)
complex(kind= 4 8)	logical(kind= 1 2 4 8)
double precision	real(kind= 4 8)

Numeric and Logical Intrinsic Functions

abs(integer real complex)	int(integer real complex)
aimag(complex)	logical(logical)
aint(real)	max(integer real)
anint(real)	min(integer real)
ceiling(real)	mod(integer real)
cmplx(real)	modulo(integer real)
cmplx(real,real)	nint(real)
conjg(complex)	real(integer real complex)
dim(integer real)	sign(integer real)
floor(real)	

Mathematical Intrinsic Functions

acos(real)	log(real complex)
asin(real)	log10(real)
atan(real)	sin(real complex)
atan2(real,real)	sinh(real)
cos(real complex)	sqrt(real complex)
cosh(real)	tan(real)
exp(real complex)	tanh(real)

Numeric Inquiry Intrinsic Functions

bit_size(integer)	precision(real complex)
digits(integer real)	radix(integer real)
epsilon(real)	range(integer real complex)
huge(integer real)	selected_int_kind(integer)
maxexponent(real)	selected_real_kind(integer,integer)
minexponent(real)	tiny(real)

Bit Manipulation Intrinsic Functions

btest(integer)	ishft(integer)
iand(integer)	ishftc(integer)
ibclr(integer)	leadz(integer)
ibits(integer)	mvbits(integer)
ibset(integer)	not(integer)
ieor(integer)	popcnt(integer)
ior(integer)	poppar(integer)

CUDA Synchronization and Fence Functions

syncthreads()	
syncthreads_count(integer logical)	
syncthreads_and(integer logical)	
syncthreads_or(integer logical)	
threadfence()	
threadfence_block()	
threadfence_system()	

Reduction Intrinsic Functions

all(logical)	minloc(integer real)
any(logical)	minval(integer real)
count(logical)	maxloc(integer real)
maxval(integer real)	sum(integer real complex)
product(integer real complex)	

Random Number Intrinsic Functions

random_number(real)
random_seed(integer)

CUDA Warp Vote Functions

allthreads(logical)
anythread(logical)
ballot(integer logical)

CUDA Atomic Functions

atomicadd	atomicmax
atomicand	atomicmin
atomiccas	atomicor
atomicdec	atomicsub
atomicexch	atomicxor
atomicinc	

CUDA Device libm Routines

use libm	
acosh[f](x)	llrint[f](x)
asinh[f](x)	lrint[f](x)
asinf(x)	llround[f](x)
acosh(x)	lround[f](x)
atan2f(x,y)	logb[f](x)
atanh[f](x)	log10f(x)
cbirt[f](x)	logf(x)
ceil[f](x)	log1p[f](x)
copysign[f](x,y)	log2[f](x)
cosf(x)	modf[f](x,y)
coshf(x)	nearbyint[f](x)
erf[f](x)	nextafter[f](x,y)
erfc[f](x)	pow[f](x,y)
expm1[f](x)	remainder[f](x,y)
expf(x)	remquo[f](x,y,i)
exp10[f](x)	rint[f](x)
exp2[f](x)	scalbn[f](x,n)
fabs[f](x)	scalbln[f](x,n)
floor[f](x)	sinf(x)
fma[f](x,y,z)	sinhf(x)
fmax[f](x,y)	sqrtf(x)
fmin[f](x,y)	tanf(x)
frexp[f](x,y)	tanhf(x)
ilogb[f](x)	tgamma[f](x)
ldexp[f](x,y)	trunc[f](x)
lgamma[f](x)	

CUDA Device Built-in Routines

use cudadevice	
__mulhi(i, j)	__double2[ui]int_r[nlzluld](r)
__mul64hi(i, j)	__u]int2double_r[nlzluld](i)
__int_as_float(i)	__double2[ui]ll_r[nlzluld](r)
__float_as_int(i)	__u]ll2double_r[nlzluld](i)
__saturatef(r)	__fadd_r[nlzluld](a, b)
__[u]sad(i, j, k)	__fmul_r[nlzluld](a, b)
__fdividef(r, d)	__fmaf_r[nlzluld](a, b, c)
fdivide[f](r, d)	__frcp_r[nlzluld](a)
__sinf(r)	__fsqrt_r[nlzluld](a)
__cosf(r)	__fdiv_r[nlzluld](a,b)
__tanf(r)	__dadd_r[nlzluld](x, y)
__expf(r)	__dmul_r[nlzluld](x, y)
__exp10f(r)	__dfma_r[nlzluld](x, y, z)
__log2f(r)	__drcp_r[nlzluld](x)
__log10f(r)	__dsqrt_r[nlzluld](x)
__logf(r)	__ddiv_r[nlzluld](x, y)
__powf(r,d)	__float2[ui]int_r[nlzluld](r)
__clz[ll](i)	__u]int2float_r[nlzluld](i)
__ffs[ll](i)	__float2[ui]ll_r[nlzluld](r)
__popc[ll](i)	__u]ll2float_r[nlzluld](i)
__brev[ll](i)	__float2half_r[nlzluld](r)
__half2float(i)	

Host Control Code

Kernel Launch

call kernel<<<grid,block[, [nbytes*]
[,streamid]]>>> (arguments)

- grid and block are integer expressions, or type(dim3) variables
- nbytes specifies how many bytes of memory to allocate for dynamic shared memory
- streamid is a stream identifier

CUF Kernel

!\$cuf kernel do[(n)] [<<< grid, block >>>]
do i_n ...
...
do i_1 ...

- Create a kernel from the following n nested loops
- grid and block are lists of n integer expressions, corresponding to the n loops, starting with the innermost
- If any expression has the value 1, that loop will not correspond to a block or thread index
- If any expression is *, the compiler will choose a size to use for that dimension

API Routines for Device Management

cudaChooseDevice(devnum, prop)
cudaDeviceGetCacheConfig(cacheconfig)
cudaDeviceGetLimit(val, limit)
cudaDeviceReset()
cudaDeviceSetCacheConfig(cacheconfig)
cudaDeviceSetLimit(limit, val)
cudaDeviceSynchronize()
cudaGetDevice(devnum)
cudaGetDeviceCount(numdev)
cudaGetDeviceProperties(prop, devnum)
cudaSetDevice(devnum)
cudaSetDeviceFlags(flags)
cudaSetValidDevices(devices, numdev)

API Routines for Thread Management

cudaThreadExit()
cudaThreadSynchronize()

API Routines for Memory Management and Data Transfer

cudaFree(devptr)
cudaFreeArray(arrayptr)
cudaFreeHost(hostptr)
cudaGetSymbolAddress(devptr, symbol)
cudaGetSymbolSize(size, symbol)
cudaHostAlloc(hostptr, size, flags)
cudaHostGetDevicePointer(devptr, hostptr, flags)
cudaHostGetFlags(flags, hostptr)
cudaHostRegister(hostptr, count, flags)
cudaHostUnregister(hostptr)
cudaMalloc(devptr, count)
cudaMallocArray(arrayptr, channeldesc, width, height)
cudaMallocHost(hostptr, size)
cudaMallocPitch(devptr, pitch, width, height)
cudaMalloc3D(pitchptr, cext)
cudaMalloc3DArray(carray, cdsc, cext)
cudaMemcpy(dst, src, count, kdir)
cudaMemcpyToArray(dsta, dstx, dsty, src, srcx, srcy, count, kdir)
cudaMemcpyAsync(dst, src, count, kdir, stream)
cudaMemcpyFromArray(dst, src, srcx, srcy, count, kdir)
cudaMemcpyFromSymbol(symbol, src, count, offset, kdir)
cudaMemcpyFromSymbolAsync(dst, symbol, count, offset, kdir, stream)
cudaMemcpyPeer(dst, dstdev, src, srcdev, count)
cudaMemcpyPeerAsync(dst, dstdev, src, srcdev, count, stream)
cudaMemcpyToArray(dsta, dstx, dsty, src, count, kdir)
cudaMemcpyToSymbol(symbol, src, count, offset, kdir)
cudaMemcpyToSymbolAsync(symbol, src, count, offset, kdir, stream)