cudaMemcpyToArray(dsta, dstx, dsty, src, count, kdir)
cudaMemcpyToSymbol(symbol, src, count, offset, kdir)
cudaMemcpyToSymbolAsync(symbol, src, count, offset, kdir, stream)
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)
cudaMemsetAsync(devptr, value, count, stream)
cudaMemset2DAsync(devptr, pitch, value, width, height, stream)
cudaMemset3DAsync(pitchptr, value, cext, stream)

API Routines for Execution Control

*cudaFuncGetAttributes(attr, func)
cudaFuncSetCacheConfig(func, cacheconfig)
cudaFuncSetSharedMemConfig(func, config)
cudaSetDoubleForDevice(d)
cudaSetDoubleForHost(d)

API Routines for Stream Management

cudaStreamCreate(stream)
*cudaStreamCreateWithFlags(stream, flags)
*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)

* These API routines are supported in device code as well as host code.

Compiler Options

```
pgfortran a.cuf -Mcuda[=options]
```

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

- `cc13|cc1x|cc20|cc2x|cc30,cc35|cc3x`
- `cuda3.2|cuda4.0|cuda4.1|cuda4.2|cuda5.0|cuda5.5`
- `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 == 1) 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
```

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 mathematical 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.

The Portland Group
www.pgroup.com/cudafortran

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

texture

- accessible read-only in device subprograms
- must be an F90 pointer, real or integer

pinned

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

managed (CUDA 6 or later only)

- declared on the host
- can be accessed from either host or device

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

Numeric and Logical Intrinsic Functions

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

Mathematical Intrinsic Functions

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

Numeric Inquiry Intrinsic Functions

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

Bit Manipulation Intrinsic Functions

btest(integer)
iand(integer)
ibclr(integer)
ibits(integer)
ibset(integer)
ieor(integer)
ior(integer)
ishft(integer)
ishftc(integer)
leadz(integer)
mvbits(integer)
not(integer)
popcnt(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)
any(logical)
count(logical)
maxval(integer | real)
minloc(integer | real)
minval(integer | real)
maxloc(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 Warp Shuffle Functions

__shfl(integer | real, [size])
__shfl_up(integer | real, [size])
__shfl_down(integer | real, [size])
__shfl_xor(integer | real, [size])

CUDA Atomic Functions

atomicadd(integer | real)
atomicand(integer)
atomiccas(integer | real)
atomicdec(integer)
atomicexch(integer | real)
atomicinc(integer)
atomicmax(integer | real)
atomicmin(integer | real)
atomicor(integer)
atomicsub(integer | real)
atomicxor(integer)

CUDA Device libm Routines

use libm

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

CUDA Device Built-in Routines

use cudadevice

__[u]mulhi(i, j)
__[u]mul64hi(i, j)
__int_as_float(i)
__float_as_int(i)
__saturatef(r)
__[u]sad(i, j, k)
__fdividef(r, d)
fdivide[f](r, d)
__sinf(r)
__cosf(r)
__tanf(r)
__expf(r)
__exp10f(r)
__log2f(r)
__log10f(r)
__logf(r)
__powf(r,d)
__clz[ll](i)
__ffs[ll](i)
__popc[ll](i)
__brev[ll](i)
__half2float(i)
__double2[u]int_r[n|z|u|d](r)
__[u]int2double_r[n|z|u|d](i)
__double2[u]ll_r[n|z|u|d](r)
__[u]ll2double_r[n|z|u|d](i)
__fadd_r[n|z|u|d](a, b)
__fmul_r[n|z|u|d](a, b)
__fmaf_r[n|z|u|d](a, b, c)
__frcp_r[n|z|u|d](a)
__fsqrt_r[n|z|u|d](a)
__fdiv_r[n|z|u|d](a,b)
__dadd_r[n|z|u|d](x, y)
__dmul_r[n|z|u|d](x, y)
__dfma_r[n|z|u|d](x, y, z)
__drcp_r[n|z|u|d](x)
__dsqrt_r[n|z|u|d](x)
__ddiv_r[n|z|u|d](x, y)
__float2[u]int_r[n|z|u|d](r)
__[u]int2float_r[n|z|u|d](i)
__float2[u]ll_r[n|z|u|d](r)
__[u]ll2float_r[n|z|u|d](i)
__float2half_r[n|z|u|d](r)

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)
cudaDeviceGetSharedMemConfig(config)
cudaDeviceReset( )
cudaDeviceSetCacheConfig(cacheconfig)
cudaDeviceSetLimit(limit, val)
cudaDeviceSetSharedMemConfig(config)
*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, cdesc, cext)
cudaMemcpy(dst, src, count, kdir)
cudaMemcpyArrayToArray(dsta, dstx, dsty, srca, srcx, srcy, count, kdir)
*cudaMemcpyAsync(dst, src, count, kdir, stream)
cudaMemcpyFromArray(dst, srca, srcx, srcy, count, kdir)
cudaMemcpyFromSymbol(symbol, src, count, offset, kdir)
cudaMemcpyFromSymbolAsync(dst, symbol, count, offset, kdir, stream)
cudaMemcpyPeer(dst, dstdev, src, srcdvc, count)
cudaMemcpyPeerAsync(dst, dstdev, src, srcdev, count, stream)