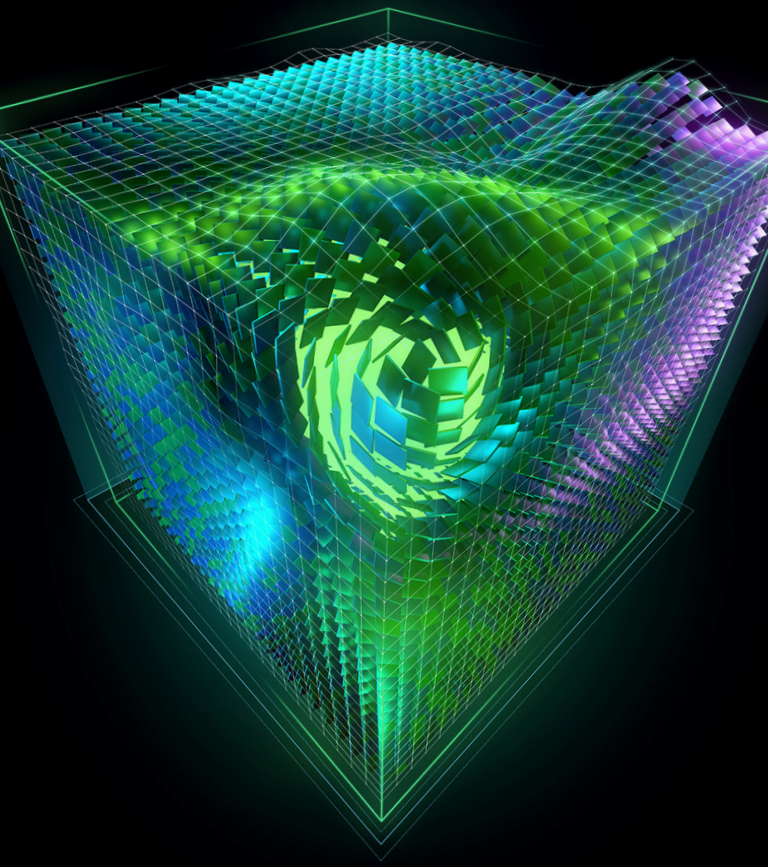
Quick Reference Guide

CUDA Fortran



PGI®
COMPILERS & TOOLS

CUDA Fortran Quick Reference Guide

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.

Subprogram Qualifiers

attributes(global), attributes(grid_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
- **grid_global** allows grid-wise synchronization; requires **cc60** or higher

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

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
- explicit: declared with **texture**, **pointer** attributes at module scope; real or integers
- implicit: declare kernel arguments with **intent(in)**; all intrinsic types (**cc35** or higher)

pinned

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

managed

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

value

- scalar dummy kernel arguments that reside on the host can be passed to kernels by using the **`value`** attribute

Predefined Types, Constants and Variables

Parameters

cuda_count_kind (count)
cuda_stream_kind (stream)

Types

C_DEVPTR (devptr)
DIM3 (dim3)
cudaDeviceProp (prop)
cudaEvent (event)
cudaExtent (cext)
cudaPos
cudaSymbol (symbol)
cudaArrayPtr (carray)
cudaChannelFormatDesc (cdesc)
cudaFuncAttributes (attr)
cudaMemcpy3DParms (p)
cudaMemcpy3DPeerParms (p)
cudaPitchedPtr (pitchptr)
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= 2* | 4 | 8)

*(kind=2 requires **`cc60`** or higher)

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
acosh
asin
asinh
atan
atanh
atan2
bessel_j0
bessel_j1
bessel_jn
bessel_y0
bessel_y1
bessel_yn
cos
cosh
exp
erf
erfc
gamma
hypot
log
log_gamma
log10
sin
sinh
sqrt
tan
tanh

Numeric Inquiry Intrinsic Functions

bit_size
digits
epsilon
huge
maxexponent
minexponent
precision
radix
range
selected_int_kind
selected_real_kind
tiny

Bit Manipulation Intrinsic Functions

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

CUDA Synchronization and Fence Functions

syncthreads
syncthreads_count
syncthreads_and
syncthreads_or
syncwarp
threadfence
threadfence_block
threadfence_system

Reduction Intrinsic Functions

all
any
count
maxloc
maxval
minloc
minval
product
sum

CUDA Warp Vote and Match Functions

activemask
all_sync
allthreads
any_sync
anythread
ballot
ballot_sync
match_all_sync
match_any_sync

CUDA Warp Shuffle Functions

__shfl
__shfl_up
__shfl_down
__shfl_xor

CUDA Atomic Functions

atomicadd
atomicand
atomiccas
atomicdec
atomicexch
atomicinc
atomicmax
atomicmin
atomicor
atomicsub
atomicxor

CUDA Device libm Routines

use libm

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

CUDA Device Built-in Routines

use cudadevice

__brev[ll]
clock
clock64
__clz[ll]
__cosf
__dadd_r[n|z|u|d]
__ddiv_r[n|z|u|d]
__dfma_r[n|z|u|d]
__dmul_r[n|z|u|d]
__double2[u]int_r[n|z|u|d]
__double2float_r[n|z|u|d]
__double2hiint
__double2loint
__drcp_r[n|z|u|d]
__dsqrt_r[n|z|u|d]
__exp10f
__expf
__fadd_r[n|z|u|d]
__fdiv_r[n|z|u|d]
fdivide[f]
__fdividef
__ffs[ll]
__float_as_int
__float2[u]int_r[n|z|u|d]
__float2[u]ll_r[n|z|u|d]
__float2half_rn
__fma_r[n|z|u|d]
__fmaf_r[n|z|u|d]
__fmul_r[n|z|u|d]
__frcp_r[n|z|u|d]
__fsqrt_r[n|z|u|d]
__half2float
__hiloint2double
__int_as_float
__[u]int2double_r[n|z|u|d]
__[u]int2float_r[n|z|u|d]
__[u]ll2double_r[n|z|u|d]
__[u]ll2float_r[n|z|u|d]
__log10f
__log2f
__logf
__longlong_as_double
__[u]mul24hi
__[u]mulhi
__popc[ll]
__powf
__[u]sad
__saturatef
__sinf
__tanf

Host Control Code

Kernel Launch

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

- **`grid`** and **`block`** are integer expressions, or type(dim3) variables

For **`grid_global`** launches the wildcard "*" can be used for the **`grid`** parameter, where the compiler will determine the number of blocks to use.

- **`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
*cudaDeviceGetCacheConfig
*cudaDeviceGetLimit
cudaDeviceGetSharedMemConfig
cudaDeviceGetStreamPriorityRange
cudaDeviceReset
cudaDeviceSetCacheConfig
cudaDeviceSetLimit
cudaDeviceSetSharedMemConfig
*cudaDeviceSynchronize
*cudaGetDevice
*cudaGetDeviceCount
*cudaGetDeviceProperties
cudaSetDevice
cudaSetDeviceFlags
cudaSetValidDevices

API Routines for Memory Management and Data Transfer

*cudaFree
cudaFreeArray
cudaFreeHost
cudaHostAlloc
cudaHostGetDevicePointer
cudaHostGetFlags
cudaHostRegister
cudaHostUnregister
*cudaMalloc
cudaMallocManaged
cudaMemcpy[Peer][Async]
cudaMemcpy2D[FromArray | ToArray][Async]
cudaMemcpy2DArraytoArray
cudaMemcpy3D

*cudaMemcpy3D[Peer][Async]
cudaMemGetInfo
cudaMemPrefetchAsync
cudaMemset
cudaMemset[2D | 3D][Async]
cudaPointerGetAttributes

API Routines for Execution Control

*cudaFuncGetAttributes
cudaFuncSetAttributes
cudaFuncSetCacheConfig
cudaFuncSetSharedMemConfig
cudaSetDoubleForDevice
cudaSetDoubleForHost

API Routines for Stream Management

cudaforSetDefaultStream
cudaforGetDefaultStream
cudaStreamAttachMemAsync
cudaStreamCreate
*cudaStreamCreateWithFlags
cudaStreamCreateWithPriority
*cudaStreamDestroy
cudaStreamGetPriority
cudaStreamQuery
cudaStreamSynchronize
*cudaStreamWaitEvent

API Routines for Event Management

cudaEventCreate
*cudaEventCreateWithFlags
*cudaEventDestroy
cudaEventElapsedTime
cudaEventQuery
*cudaEventRecord
cudaEventSynchronize

API Routines for Occupancy

cudaOccupancyMaxActiveBlocksPerMultiprocessor
cudaOccupancyMaxActiveBlocksPerMultiprocessorWithFlags

API Routines for Unified Addressing and Peer Device Memory Access

cudaDeviceCanAccessPeer
cudaDeviceDisablePeerAccess
cudaDeviceEnablePeerAccess
cudaMemAdvise
cudaPointerGetAttributes

API Routines for Error Handling

*cudaGetErrorString
*cudaGetLastError
*cudaPeekAtLastError

API Routines for Version Management

cudaDriverGetVersion
*cudaRuntimeGetVersion

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

Pre-compiled Host Modules

- **cudafor** contains Fortran interfaces for the CUDA API routines listed in this guide, the predefined types, and Fortran interfaces for device versions of the reduction intrinsics sum, maxval, and minval.
- **cublas** contains Fortran interfaces for the CUBLAS library. The standard Fortran BLAS interfaces are also overloaded to accept device data. Link with **-Mcudalib=cublas**.
- **cufft** contains Fortran interfaces for the CUFFT library. Link with **-Mcudalib=cufft**.
- **curand** contains Fortran interfaces for the CURAND library. Link with **-Mcudalib=curand**.
- **cusolver** contains Fortran interfaces for LAPACK-like routines Link with **-Mcudalib=cusolver**.
- **cusparse** contains Fortran interfaces for the CUSPARSE library. Link with **-Mcudalib=cusparse**.

See pgicompilers.com/docs/pgicudaint for additional information

Pre-compiled Device Modules

- **cudadevice** allows access and interfaces to many of the CUDA device built-in routines. Used by default.
- **libm** provides interfaces to standard libm functions which are not in the Fortran intrinsic library.
- **wwma** interfaces to warp matrix multiply and accumulate (WMMA) units (aka Tensor Cores) that operate on half-precision (**real(2)**) multiplicands. Requires **cc70** or higher.

  Types: **WMMASubMatrix**

  Functions: **wmmaLoadMatrix(), wmmaStoreMatrix(), wmmaMatMul()**

- **cooperative_groups** provides interfaces to cooperative group functionality.

  Types: **grid_group, coalesced_group, thread_group (grid_group** requires **cc60** or higher**)**

  Functions: **this warp( )**, **this_grid( )**, **this_thread_block( )** (**this_grid( )** requires **cc60** or higher)

  syncthreads( ) is overloaded to synchronize across specified groups:
  call **syncthreads(gg | cg | tg)**

  shfl_sync( ) shfl functions

Compiler Options

pgfortran a.cuf -Mcuda[=options]

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

cc30	Compile for compute capability 3.0
cc35	Compile for compute capability 3.5
cc50	Compile for compute capability 5.0
cc60	Compile for compute capability 6.0
cc70	Compile for compute capability 7.0
cc75	Compile for compute capability 7.5
ccall	Compile for all supported compute capabilities
cudaX.Y	Use CUDA X.Y Toolkit compatibility, where installed
[no]debug	Generate GPU debug information
[no]lineinfo	Generate GPU line information
fastmath	Use fast math library
[no]flushz	Enable flush-to-zero mode on the GPU
keepgpu	Keep kernel source files
[no]llvm	Deprecated; see [no]nvvm
[no]nvvm	Use libNVVM to generate device code
keepbin	Keep CUDA binary files
keepptx	Keep PTX portable assembly files
madconst	Put Module Array Descriptors in CUDA Constant Memory
maxregcount:<n>	Set maximum number of registers to use on the GPU
charstring	Enable limited support for character strings in GPU kernels
[no]fma	Generate fused mul-add instructions (default at -O3)
loadcache	Choose what hardware level cache to use for global memory loads

L1	Use L1 cache
L2	Use L2 cache
[no]unroll	Enable automatic inner loop unrolling (default at -O3)
ptxinfo	Print informational messages from PTXAS
[no]rdc	Generate relocatable device code

Enter **pgfortran -Mcuda -help** for a complete list of compiler options.

See also the PGI Fortran CUDA Library Interfaces document

www.pgicompilers.com/docs/pgicudaint

PGI®
COMPILERS &TOOLS

www.pgicompilers.com/cudafortran