CUDA C Language Extensions

Function Type Qualifiers

`__device__` executed on the device, callable from the device only

`__global__` declares a function as being a kernel, executed on the device, callable from the host, must have `void` return type

`__host__` executed on the host, callable from the host only

It is equivalent to declare a function with only the `__host__` qualifier or to declare it without any of the `__host__`, `__device__`, or `__global__` qualifier. In either case the function is compiled for the host only. The `__global__` and `__host__` qualifiers cannot be used together. The `__device__` and `__host__` qualifiers can be used together however, in which case the function is compiled for both the host and the device.

Variable Type Qualifiers

`__device__` declares a variable that resides on the device

`__constant__` declares a variable that resides in constant memory space, has the lifetime of an application, is accessible from all the threads within the grid and from the host through the runtime library

`__shared__` declares a variable that resides in the shared memory space of a thread block, has the lifetime of the block, is only accessible from all the threads within the block

Built-in Variables

Variables that are only valid within functions that are executed on the device.

`gridDim` variable of type `dim3` (i.e. `struct` comprising three `int`s) containing the dimensions of the grid

`blockIdx` variable of type `uint3` containing the block index within the grid

`blockDim` variable of type `dim3` containing the dimensions of the block (i.e., the number of threads in each direction)

`threadIdx` variable of type `uint3` containing the thread index within a block

`warpSize` variable of type `int` containing the warp size in threads

Dynamic Memory Allocations

Host	Device	cudaSetDeviceFlags	Annotation
cudaMallocHost cudaFreeHost	cudaMalloc cudaFree		page-locked
cudaMallocHost cudaFreeHost	cudaHostGetDevicePointer	cudaDeviceMapHost	zero-copy
cudaMallocHost cudaFreeHost	cudaMallocPitch cudaFree		2D-array
new delete	cudaMalloc cudaFree		
cudaMemcpyToSymbol			constant memory

page-locked (i.e. pinned): highest bandwidth between host and device
zero-copy: performance gain on integrated GPUs
2D-array: pitched row sizes in order to match warp size (pitch, stride)
constant memory constant static variables in the device's constant memory

Tips

High Priority

1. Avoid different execution paths within the same warp.
2. Avoid the use of `__syncthreads` inside divergent code.
3. Ensure global memory accesses are coalesced whenever possible.
4. Minimize data transfer between the host and the device, even if it means running some kernels on the device that do not show performance gains when compared with running them on the host CPU.
5. Minimize the use of global memory. Prefer shared memory access where possible.
6. To get the maximum benefit from CUDA, focus first on finding ways to parallelize sequential code.
7. To maximize developer productivity, profile the application to determine hotspots and bottlenecks.
8. Use the effective bandwidth of your computation as a metric when measuring performance and optimization benefits.

Medium Priority

1. Prefer faster, more specialized math functions over slower, more general ones when possible.
2. The number of threads per block should be a multiple of 32 threads, because this provides optimal computing efficiency and facilitates coalescing.
3. To hide latency arising from register dependencies, maintain sufficient numbers of active threads per multiprocessor (i.e., sufficient occupancy).
4. Use shared memory to avoid redundant transfers from global memory.
5. Use signed integers rather than unsigned integers as loop counters.

6. Use the fast math library whenever speed trumps precision.

Low Priority

1. Avoid automatic conversion of doubles to floats.
2. Make it easy for the compiler to use branch predication in lieu of loops or control statements.
3. Use shift operations to avoid expensive division and modulo calculations.
4. Use zero-copy operations on integrated GPUs for CUDA Toolkit version 2.2 and later.

CUDA Library Elements to be used (an outline)

Pragmas and Macros

```
#pragma unroll
```

`__NVCC__`	defined when compiling C/C++/CUDA source files
`__CUDACC__`	defined when compiling CUDA source files
`__CUDACC_VER__`	defined with the full version number of `nvcc`

Types

```
cudaComputeMode
cudaDeviceProp
cudaError
cudaEvent_t
cudaMemcpyKind
```

Values

```
CUDART_VERSION
cudaDeviceMapHost
cudaHostAllocMapped
cudaMemcpyDeviceToHost
cudaMemcpyHostToDevice
cudaSuccess
```

Functions

```
cudaDeviceReset
cudaDeviceSynchronize
cudaDriverGetVersion
cudaEventCreate
cudaEventDestroy
cudaEventElapsedTime
cudaEventRecord
cudaEventSynchronize
cudaFree
cudaFreeHost
```

cudaGetDeviceCount
cudaGetDeviceProperties
cudaGetErrorString
cudaGetLastError
cudaHostGetDevicePointer
cudaMalloc
cudaMallocHost
cudaMallocPitch
cudaMemcpy
cudaMemcpy2D
cudaMemcpyToSymbol
cudaMemset
cudaRuntimeGetVersion
cudaSetDevice
cudaSetDeviceFlags
__syncthreads