CUDA C Programming Quick Reference[1]

Function Qualifiers

`__global__`	called from host, executed on device
`__device__`	called from device, executed on device (always inline when Compute Capability is 1.x)
`__host__`	called from host, executed on host
`__host__ __device__`	generates code for host and device
`__noinline__`	if possible, do not inline
`__forceinline__`	force compiler to inline

Variable Qualifiers (Device)

`__device__`	variable on device (Global Memory)
`__constant__`	variable in Constant Memory
`__shared__`	variable in Shared Memory
`__restrict__`	restricted pointers, assert to the compiler that pointers are not aliased (cf. aliased pointer)
– No Qualifier –	automatic variable, resides in Register or in Local Memory in some cases (local arrays, register spilling)

Built-in Variables (Device)

dim3 gridDim	dimensions of the current grid (gridDim.x, . . .) (composed of independent blocks)
dim3 blockDim	dimensions of the current block (composed of threads) (total number of threads should be a multiple of warp size)
uint3 blockIdx	block location in the grid (blockIdx.x, . . .)
uint3 threadIdx	thread location in the block (threadIdx.x, . . .)
int warpSize	warp size in threads (instructions are issued per warp)

Shared Memory

Static allocation	`__shared__ int a[128]`
Dynamic allocation (at kernel launch)	`extern __shared__ float b[]`

Host / Device Memory

Allocate pinned / page-locked Memory on host	cudaMallocHost(&dptr, size) (for higher bandwidth, may degrade system performance)
Allocate Device Memory	cudaMalloc(&devptr, size)
Free Device Memory	cudaFree(devptr)
Transfer Memory	cudaMemcpy(dst, src, size, cudaMemcpyKind kind) kind = {cudaMemcpyHostToDevice, . . . }
Nonblocking Transfer	cudaMemcpyAsync(dst, src, size, kind[, stream]) (host memory must be page-locked)
Copy to constant or global memory	cudaMemcpyToSymbol(symbol, src, size[, offset[, kind]]) kind=cudaMemcpy[HostToDevice|DeviceToDevice]

Synchronizing

Synchronizing one Block	`__syncthreads()` (device call)
Synchronizing all Blocks	cudaDeviceSynchronize() (host call, CUDA Runtime API)

Kernel

Kernel Launch	kernel<<<dim3 blocks, dim3 threads[, ...]>>>(arguments)

CUDA Device Management

Init device (context)	cudaSetDevice(devID)
Reset current device	cudaDeviceReset()

CUDA Runtime API Error Handling

CUDA Runtime API error as String	cudaGetErrorString(cudaError_t err)
Last CUDA error produced by any of the runtime calls	cudaGetLastError()

OpenGL Interoperability

Init device (within OpenGL context)	cudaGLSetGLDevice(devID) (mutually exclusive to cudaSetDevice())
Register buffer object (must not be bound by OpenGL)	cudaGraphicsGLRegisterBuffer(&res, id, flags) `Res:` cudaGraphicsResource pointer `id:` OpenGL Buffer Id `flags:` register flags (read/write access)
Register texture or render buffer	cudaGraphicsGLRegisterImage(&res, id, target, flags)

Graphics Interoperability

Unregister graphics resource	cudaGraphicsUnregisterResource(res)
Map graphics resources for access by CUDA	cudaGraphicsMapResources(count, &res[, stream])
Get device pointer (access a mapped graphics resource) (OpenGL: buffer object)	cudaGraphicsResourceGetMappedPointer(&dptr, size, res)
Get CUDA array of a mapped graphics resource (OpenGL: texture or renderbuffer)	cudaGraphicsSubResourceGetMappedArray(&a, res, i, lvl)
Unmap graphics resource	cudaGraphicsUnmapResources(count, &res[, stream])

CUDA Texture

Textures are read-only global memory, but cached on-chip, with texture interpolation

Declare texture (at file scope)	texture<DataType,TexType,Mode> texRef
Create channel descriptor	cudaCreateChannelDesc<DataType>()
Bind memory to texture	cudaBindTexture(offset, texref, dptr, channelDesc, size)
Unbind texture	cudaUnbindTexture(texRef)
Fetch Texel (texture pixel)	tex1D(texRef, x) tex2D(texRef, x, y) tex3D(texRef, x, y, z) tex1DLayered(texRef, x, layer) tex2DLayered(texRef, x, y, layer)

CUDA Streams (Concurrency Management)

Stream = instruction sequence. Streams may execute their commands out of order.

Create CUDA Stream	cudaStreamCreate(cudaStream_t &stream)
Destroy CUDA Stream	cudaStreamDestroy(stream)
Synchronize Stream	cudaStreamSynchronize(stream)
Stream completed?	cudaStreamQuery(stream)

[1] Incomplete Reference for CUDA Runtime API. July 5, 2012. Contact: wmatthias@t-online.de. Cf. Complete Reference: "NVIDIA CUDA C Programming Guide", Version 4.0

Technical Specifications

Compute Capability	1.0	1.1	1.2	1.3	2.x	3.0
Max. dimensionality of grid	2				3	
Max. dimensionality of block	3					
Max. x-,y- or z-dimension of a grid	$2^{16}-1$					$2^{32}-1$
Max. x- or y-dimension of a block	512				1024	
Max. z-dimension of a block	64					
Max. threads per block	512				1024	
Warp Size	32					
Max. resident blocks per SM	8					16
Max. resident warps per SM	24		32		48	64
Max. resident threads per SM	768		1024		1536	2048
Number of 32-bit registers per SM	8K		16K		32K	64K
Max. registers per thread	10 (+1)		16 (+1)		63 (+1)	
Max. shared memory per SM ($\geq$2.0: configurable L1 Cache)	16 KB				48KB or 16KB	
Number of shared memory banks	16				32	
Constant memory size	64 KB					
Local memory per thread	16 KB				512 KB	
Cache working set per SM for constant	8 KB					
Cache working set per SM for texture	device dependent, 6-8 KB					
Max. instructions per kernel	2 million					512 million
Max. width for 1D texture (CUDA array)	8192				65536	
Max. width 1D texture (linear memory)	2^{27}					
Max. width×layers 1D texture	8192×512				16384×2048	
Max. width×height for 2D texture	65536×32768				65536×65535	
Max. width×height×layers for 2D texture	$8192 \times 8192 \times 512$				$16384 \times 16384 \times 2048$	
Max. width×height×depth 3D texture	2048^3					4096^3
Max. textures bound to kernel	128					256
Max. width for 1D surface	N/A				65536	
Max. width×height for 2D surface	N/A				65536×32768	
Max. surfaces bound to kernel	N/A				8	16

Architecture Specifications

Compute Capability	1.0	1.1	1.2	1.3	2.0	2.1	3.0
Number of cores (with FPU and ALU)	8				32	48	192
Number of special function units	2				4	8	32
Number of texture units	2				4	8	32
Number of warp schedulers	1				2	2	4
Number of instructions issued at once by scheduler	1				1	2	2

[1]Sources: http://en.wikipedia.org/wiki/CUDA and Nvidia

Supported GPUs

CC	GPUs	Information
1.0	G80, G92, G92b, G94, G94b	*"Unified Shader Architecture"* Supporting GPU programming with C.
1.1	G86, G84, G98, G96, G96b, G94, G94b, G92, G92b	- 32-bit Integer atomic functions for global memory, ...
1.2	GT218, GT216, GT215	*"Tesla"* - Warp vote functions, ...
1.3	GT200, GT200b	- Double-Precision, ...
2.0	GF100, GF110	*"Fermi"* - ECC, Better Caches (L1 and L2), GigaThread-Engine, better atomics, dual warp, unified address space, ...
2.1	GF104, GF114, GF116, GF108, GF106	...
3.0	GK104, GK106, GK107	*"Kepler"* - Polymorph Engine 2.0, GPU Boost, TXAA, SMX (next-generation SM)
3.5	GK110 (GPGPU)	- Dynamic Parallelism, Hyper-Q, Grid Management Unit

CUDA Memory

Memory	Location	Cached	Access	Scope	Lifetime
Register	On-chip	N/A	R/W	Thread	Thread
Local	Off-chip	No*	R/W	Thread	Thread
Shared	On-chip	N/A	R/W	Block	Block
Global	Off-chip	No*	R/W	Global	Application
Constant	Off-chip	Yes	R	Global	Application
Texture	Off-chip	Yes	R	Global	Application
Surface	Off-chip	Yes	R/W	Global	Application

*) Devices with compute capability $\geq$2.0 use L1 and L2 Caches.

Occupancy

$= \frac{\#\text{active warps per SM}}{\#\text{possible warps per SM}}$ ($\nearrow$ ExcelSheet "Occupancy Calculator")

Higher occupancy $\neq$ better performance (it's just more likely to hide latencies)

Potential occupancy limiters: Register usage, Shared Memory usage, Block size

Helpful nvcc compiler flag: `--ptxas-options=-v`(show memory usage of kernel)