CUDA C Programming Quick Reference[1]

Function Qualifiers

`__global__`	called from host, executed on device
`__device__`	called from device, executed on device
`__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)
dim3 blockIdx	block location in the grid (blockIdx.x, ...)
dim3 threadIdx	thread location in the block (threadIdx.x, ...)

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<<<blocks, threadsPerBlock[, smem_size[, stream]]>>>(..)

CUDA Device Management

Init device (context)	cudaSetDevice(devID)
Reset current device	cudaDeviceReset() (for profiler and flushing)

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] November 12, 2019. Cf. Complete Reference: "NVIDIA CUDA C Programming Guide" – Note, that functions may have optional arguments not listed here

	Tesla				Fermi		Kepler			Maxwell	
Compute Capability	1.0	1.1	1.2	1.3	2.0	2.1	3.0	3.5	3.7	5.0	5.2
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			32	
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	8 K		16 K		32 K		64 K	128 K		64 K	
Max. registers per thread	124				63		255				
Max. shared memory per SM (≥2.0: configurable L1 Cache)	16 KB				48 KB			112 KB		64 KB	96 KB
Number of shared memory banks	16						32				
Local memory per thread	16 KB						512 KB				
Constant memory size	64 KB										
Cache working set per SM for constant	8 KB									10 KB	
Cache working set per SM for texture	6-8 KB				12 KB		12 KB-48 KB			24 KB	48 KB
Max. instructions per kernel	2 million						512 million				
Max. width for 1D texture (array)	8192						65 536				
Max. width 1D texture (linear)	2^{27}										
Max. width×layers for 1D texture	8192×512						16 384×2048				
Max. textures bound to kernel	128						256				
Max. width×layers for 1D surface	N/A						65 536×2048				
Max. surfaces bound to kernel	N/A				8		16				
Architecture Specifications											
Number of cores (with FPU and ALU)	8				32	48	192			128	
Number of special function units	2				4	8	32				
Number of texture units	2				4	8	16			8	
Number of warp schedulers	1				2		4				
Number of instructions issued by scheduler	1						2				
Compute Capability	1.0	1.1	1.2	1.3	2.0	2.1	3.0	3.5	3.7	5.0	5.2

CC	GPUs	Features
	Fermi	
2.0	GF100, GF110	ECC, Better Caches (L1 and L2), dual warp scheduler, concurrent kernel execution, better atomics, int64 shared memory atomics, float32-atomicAdd, unified address space, ballot, threadfence, surface, FMA, int32 ALU
2.1	GF104, ...	–
	Kepler (Focus: perf/watt, doubles, HPC)	
3.0	GK104, GK106, GK107	Polymorph Engine 2.0, GPU Boost, TXAA, warp shuffle, bindless textures, h.264 encoder NVENC, adaptive VSync, PCIe 3.0
3.5	GK110, GK208	Dynamic Parallelism, Hyper-Q, Grid Management Unit, GPUDirect (RDMA), funnel shift
3.7	GK210 (K80)	
	Maxwell (Focus: perf/watt, perf/area, single-precision, Gaming)	
5.0	GM107, GM108	
5.2	GM200, GM204, GM206	Polymorph Engine 3.0, VXGI (Global Illumination), H.265 encoding

GPU	GTX 580	GTX 680	GTX 780	GTX 980
Launch	Nov 2010	Mar 2012	May 2013	Sep 2014
Model	GF110	GK104	GK110	GM204
Core Clock (MHz)	772	1006	863	1126
Shader Clock (Mhz)	1544	–	–	–
Boost Clock (MHz)	–	1058	900	1216
PCIe Bus Support	2.0	3.0	3.0	3.0
CUDA Cores	512	1536	2304	2048
Memory Bandwidth (GB/sec)	192.4	192.2	288.4	224
Memory Clock (Mhz)	4008	6008	6008	7010
Memory Interface Width (bit)	384	256	384	256
Standard Memory Config MiB	1536	2048	3072	4096
Texture Fill Rate (billion/sec)	49.4	128.8	160.5	144
Max Temp. °C	97	98	95	98
Max Power W	244	195	250	165
SM count	16	8	12	16
Transistors	3×10^9	3.5×10^9	7×10^9	5.2×10^9
GFLOPS/s (SP/DP)	1581 / 198	3090 / 128	3977 / 166	4612 / 144
Fab-Size nm	40	28	28	28
Die Size mm^2	520	294	561	398
L2 Cache Size KiB	768	512	1536	2048
ROPs	48	32	48	64
Texture Units	64	128	192	128

Memory	Location	Cached	Access	Scope	Lifetime
Register	On-chip	N/A	R/W	Thread	Thread
Local	Off-chip	Yes	R/W	Thread	Thread
Shared	On-chip	N/A	R/W	Block	Block
Global	Off-chip	Yes	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