

CUDA/GPU intro

- GPU suited to *data parallel* computation, same program on different data elements
- This computational model maps data elements to parallel processing threads
- CUDA has three key abstractions:
 - hierarchy of thread groups
 - shared memory
 - barrier synchronization

Bill Brouwer 08/09 wbrouwer@stoneridgetechnology.com

Programming Model

- *Kernels*: Special C functions which when called are executed N times in parallel by N different CUDA threads
- defined by use of the `__global__` declaration specifier
- number of threads required given in `<<<...>>>`
- each thread given unique thread ID that is accessible in kernel via `threadIdx` variable.

Bill Brouwer 08/09 wbrouwer@stoneridgetechnology.com

Thread Hierarchy

- `threadIdx` is a 3-compt vector used to reference threads with a 1D,2D or 3D index

	Thread Index	ID	Size
1D	x	x	D_x
2D	(x,y)	(x+yD_x)	(D_x,D_y)
3D	(x,y,z)	(x+yD_x+zD_xD_y)	(D_x,D_y,D_z)

- Threads can cooperate via *shared memory*

Bill Brouwer 08/09 wbrouwer@stoneridgetechnology.com

Thread Execution I

- Execution must be synchronized via sync points in the kernel using `__syncthreads()`
- A kernel can be executed by multiple equally shaped *thread blocks*
$$total\ threads = threads/block \times blocks$$
- Multiple blocks organized into 1/2D grid
- Dimension of grid specified by first parameter in `<<<...>>>`

Bill Brouwer 08/09 wbrouwer@stoneridgetechnology.com

Thread Execution II

- Each block in grid can be identified via 1/2D index in kernel function, using `blockIdx` variable.
- Dimension of block -> `blockDim`
- Thread blocks execute independently; any order, parallel or series.
- Allows for scalable, multi-core/processor code

Bill Brouwer 08/09 wbrouwer@stoneridgetechnology.com

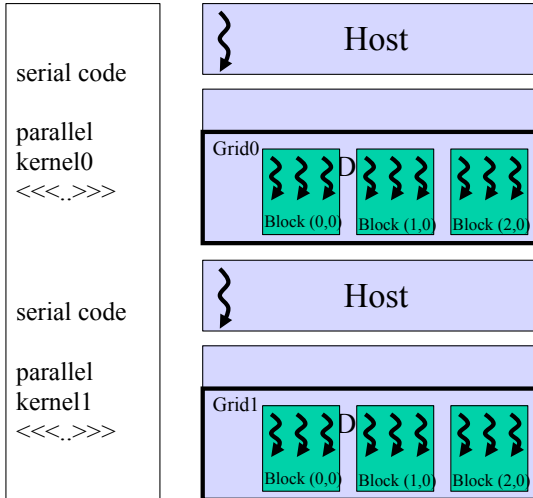
Memory I

- Threads have multiple memory options
 - per thread
 - per block
 - global
- Two additional memory resources; *constant* and *texture*
- CUDA assumes that threads may execute on physically separate device to host running C program

Bill Brouwer 08/09 wbrouwer@stoneridgetechnology.com

Memory II

C program
sequential
*.exe



Bill Brouwer 08/09 wbrouwer@stoneridgetechnology.com

Hardware I

- CUDA->scalable array of multi-threaded streaming multiprocessors (MP)
- MP's consist of 8 scalar cores, 2 special function units for transcendentals, multithreaded instruction unit and on-chip shared memory
- MP creates, manages and executes concurrent threads in groups of 32 called *warps*
- Thread management via SIMT-> *single instruction, multiple thread*
- MP maps each thread to one scalar processor core

Bill Brouwer 08/09 wbrouwer@stoneridgetechnology.com

Hardware II

- When MP receives thread block(s), split into warps and scheduled by SIMT unit
- When instructions are issued, SIMT selects available warp and directs instr. to threads in warp
- *Warps execute one common instruction at a time*
- May diverge based on conditional branches, in which case execution becomes serial or disabled until the branch is complete
- Best to coordinate threads, for obvious reasons...

Bill Brouwer 08/09 wbrouwer@stoneridgetechnology.com

Hardware III

- Each MP has on-chip memory of following types:
 - one set of local 32 bit registers per processor
 - shared parallel data cache or *shared memory* for all scalar cores
 - shared read-only *constant cache* for all scalar cores
 - shared read-only *texture cache* for all scalar cores
- Number of blocks an MP can process at once -> active blocks/MP
- Dictated by registers/thread & shared memory/block for a given kernel

Bill Brouwer 08/09 wbrouwer@stoneridgetechnology.com

C for CUDA I

- Extensions to C, targets sections of code to run on device
- A runtime library split into:
 - A host compt (on CPU)
 - A device compt (on GPU)
 - Common compt, works on device & host
- Function type qualifiers to specify whether a function executes on host or device & whether it's callable from host/device
- Variable type qualifiers to specify the memory location on the device of a variable

Bill Brouwer 08/09 wbrouwer@stoneridgetechnology.com

C for CUDA II

- Directive to specify how kernel is executed on device from host
- Four built-in variables that specify grid & block dimensions, block & thread indices
- Function type qualifiers:
 - `__device__` *executed and called from device*
 - `__global__` *executed on device, called from host*
 - `__host__` *executed and called from host*
- These have restrictions eg., `__global__` and `__host__` can't be used together

Bill Brouwer 08/09 wbrouwer@stoneridgetechnology.com

C for CUDA III

- Variable type qualifiers:

`__device__` *global mem, for all threads & host, lasts for appl*

`__constant__` *const mem, for all threads & host, lasts for appl*

`__shared__` *in thread block, for block threads, lasts for block*

- Writes to shared variables only (guaranteed) visible to threads after `__syncthreads()`
- None can be defined as external via `extern`
- `__constant__` is only assigned via host, not device
- `__device__` and `__constant__` are only allowed at file scope

Bill Brouwer 08/09 wbrouwer@stoneridgetechnology.com

Execution I

- Call to `__global__` must specify execution configuration for call
- Define dimension of the grid and blocks that will be used to execute the function on the device, and associated stream(s)
- specified by inserting:

`<<<Dg, Db, Ns, s>>>`

btwn function name and args

- `Dg` is of type `dim3`, specifies the size and dimension of the grid
- `Dg.x*Dg.y` is number of blocks

Bill Brouwer 08/09 wbrouwer@stoneridgetechnology.com

Execution II

- Db is of type `dim3`, specifies the size and dimension of each block, such that `Db.x*Db.y*Db.z` is number of threads/blocks
- (optional, defaults =0) `Ns` is of type `size_t`, no. of bytes in shared memory that is dynamically allocated/block for this call, in addition to statically allocated mem.
- (optional, defaults =0) `S` is of type `cudaStream_t`, specifies associated stream
- Examples; function declared as:

```
__global__ void Func(float* parameter);
```

must be called as:

```
Func<<< Dg, Db, Ns >>>(parameter);
```

Bill Brouwer 08/09 wbrouwer@stoneridgetechnology.com

Built-in Variables

- `gridDim` of type `dim3`, contains the dimension of the grid
- `blockIdx` is of type `uint3`, contains block index within grid
- `blockDim` is of type `dim3`, contains block dimension
- `threadIdx` is of type `uint3`, contains thread index within block
- `warpSize` is of type `int`, contains warp size in threads

Bill Brouwer 08/09 wbrouwer@stoneridgetechnology.com

Compilation with NVCC

- `nvcc` separates device & host code, and compiles device code into assembly or binary
- Generated host code output as C or object code via host compiler
- `__noinline__` -> by default the opposite takes place
- `#pragma unroll` -> by default compiler unrolls small loops with a known trip count, control via
- `#pragma unroll x` <- times to unroll

Bill Brouwer 08/09 wbrouwer@stoneridgetechnology.com

Common Runtime Compt I

- Can be used by both host and device functions.
- Some vector types -> char1, char2,... char4,
float1, ..., float4, short1,...
- Derived from basic types, these are structures where 1st,
2nd, 3rd, 4th compts accessed via fields x,y,z and w.
- Produced via constructor make_<type name> eg.,
`int2 make_int2(int x,int y)`

Bill Brouwer 08/09 wbrouwer@stoneridgetechnology.com