

An introduction to The CUDA programming environment

- Introduction : the rise of GPUs
- NVIDIA GPUS and CUDA
- Basic syntax
- Memory organization
- Examples

GPUs and the CUDA environment


- **Currently a very popular approach to: inexpensive supercomputing**
- **See a series of article titled CUDA - supercomputing for the masses by Rob Farber in 'Dr. Dobbs'**

<http://www.ddj.com/cpp/207200659>

- **You can buy a Teraflop peak power for around \$1,350.**

www.nvidia.com/object/product_tesla_C2050_C2070_us.html

GPUs and the CUDA environment

- GPUs [Graphic Processing Units] are very powerful co-processors for graphics.
- Idea: why not use them for numerical computing?
- GPUs are present in every workstation - for graphics processing
-  Find out what graphics card you have on your machine or laptop..
- Characteristics:
 - large data arrays, streaming data
 - fine-grain SIMD computations
 - single precision floating point computation

- **Difficulty: software.**
- **Solution: CUDA**
- **CUDA = Compute Unified Device Architecture**
- **Introduced in 2006 for NVIDIA GPUs**
- **Idea of attached processor [or co-processor]– Not new [e.g. FPS AP-120B ‘array processor’ unveiled in 1981]**

GPUs and the CUDA environment

- Example: machines parallel-1 - parallel-8 in ITLABS

GeForce 9800 GTX+



# cores	128 (16 PEs × 8)
Memory	512 MB
Peak rate	705 Gfl [single prec.]
Clock rate	1.84 Ghz
'Compute Capability'	1.1

Tesla C1060



- * 240 cores per GPU
- * 4 GB memory
- * Peak rate: 930 Gfl [single]
- * Clock rate: 1.3 Ghz
- * 'Compute Capability': 1.3 [allows double precision]

CUDA environment: Device and Host

- Host processor (CPU) and Device (GPU)
- Model built around many threads executed on the device

SIMT: Single Instruction Multiple Threads

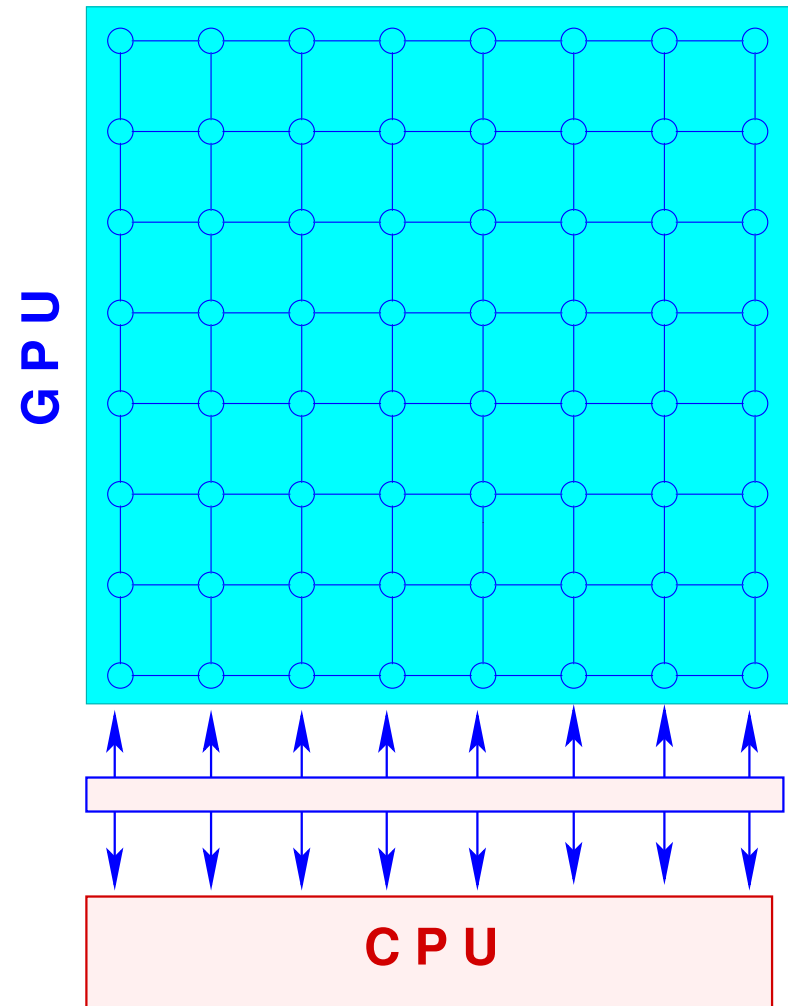
- A **Kernel** == a piece of code executed on the device
- Each kernel is run in a thread.
- Idea: generate many threads (in the form of an SIMT code) which will be run on the GPU
- Host code may be C, C++, fortran90, ..
- Kernels are in C with CUDA syntax extensions

The CUDA environment: The big picture

- A host (CPU) and an attached device (GPU)

Typical program:

1. Generate data on CPU
2. Allocate memory on GPU
`cudaMalloc(...)`
3. Send data Host → GPU
`cudaMemcpy(...)`
4. Execute GPU 'kernel':
`kernel <<<(...)>>>(...)`
5. Copy data GPU → CPU
`cudaMemcpy(...)`



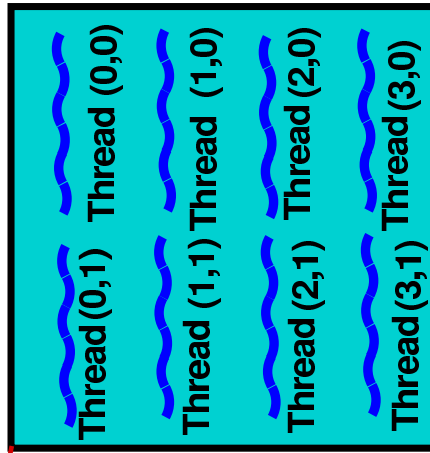
Threads, Warp, Blocks, and Grids

- A group of 32 Threads is a **Warp**
- Warps grouped into thread **blocks**
- Blocks have ≤ 512 threads
- Thread blocks are grouped into grids.

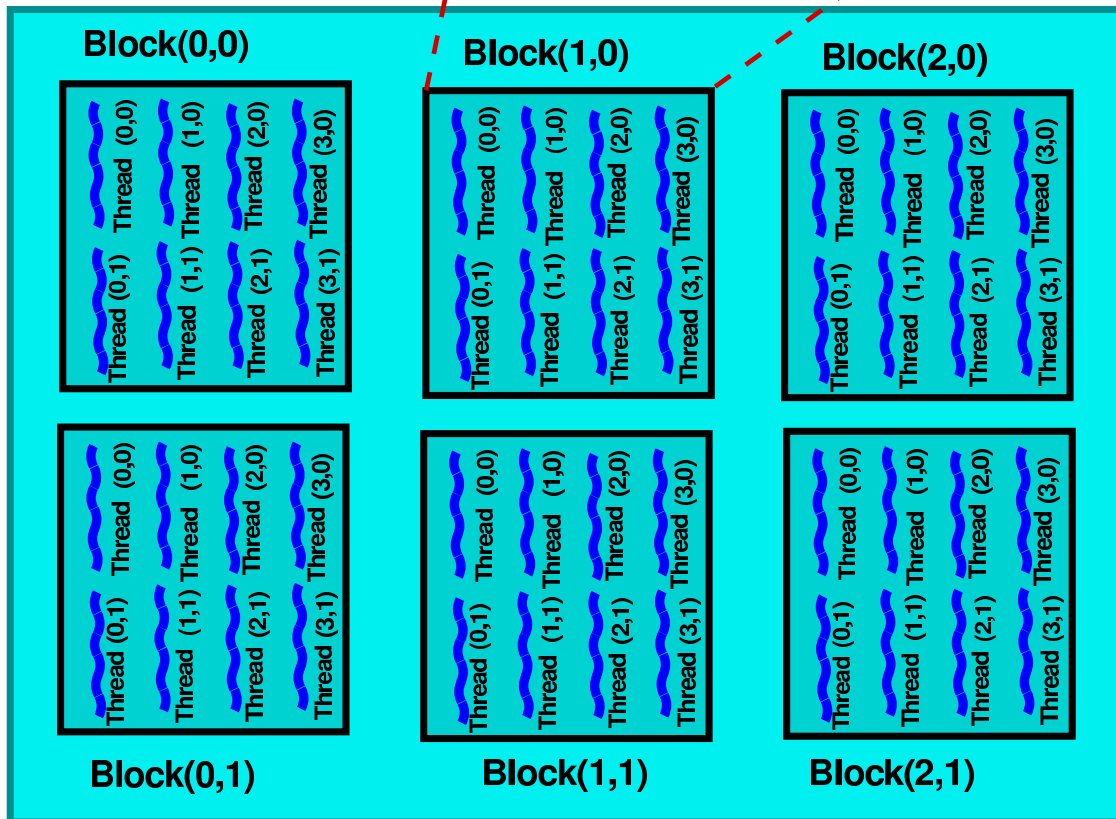
Thread → Block of Threads → Grid of Blocks

- Lots of flexibility in selecting block/grid shapes and dimensions

 Thread



BLOCK



GRID

- Blocks and grids may be 1d, 2d, or 3d
- Related kernel variables

`gridDim, blockIdx, blockDim, threadIdx`

`blockIdx, threadIdx` are 3-dimensional - can invoke

`blockIdx.x, blockIdx.y, blockIdx.z`

and

`threadIdx.x, threadIdx.y, threadIdx.z`

Function Type Qualifiers

`__device__` : declares a function which executes on device.
[Callable from the device only.]

`__global__` declares a kernel function - which is Executed on device, Callable from host only.

`__host__` declares a **host** function [executed on host, callable from host only]

If no qualifiers → considered host [but can also combine `__host__` and `__device__`]

➤ There are some restrictions – see docs. For example recursion not supported on device. ...

Example:

```
// Kernel definition:  
__global__ void vecAdd(float *x,float *y,float *z)  
{  
    int i = threadIdx.x;  
    z[i] = x[i] + y[i];  
}
```

```
int main {  
    ...  
    // Kernel call:  
    vecAdd<<<1, n>>>(xd, yd, zd);  
}
```

CUDA environment: Basic syntax

Kernels are called with the `<<< >>>` construct:

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

- Dg = dimensions of the grid (type dim3)
- Db = dimensions of the block (type dim3)
- Ns = number of bytes shared memory dynamically allocated / block (type size_t). 0 default

- What is type dim3? An integer vector type [uint3] - used to specify dimensions
- Declare as `dim3 var(dimx, dimy, dimz)`,
- ... retrieve components as `var.x`, `var.y`, `var.z`
- Unspecified components set to 1

Built-in variables

- `gridDim` is of type `dim3`. Contains dimension of grid. Similarly for `blockDim`
- Can retrieve block dimensions from `blockDim.x`, `blockDim.y`, `blockDim.z`
- `blockIdx` (type: `uint3`) contains block ID within grid
- `threadIdx` (type: `uint3`) contains thread index within block.

Example:

// Kernel definition

```
__global__ void MatAdd(float A[N][N],
    float B[N][N], float C[N][N]) {
    int i = threadIdx.x;
    int j = threadIdx.y;
    C[i][j] = A[i][j] + B[i][j];
}
int main()
{
    ...
    // Kernel invocation
    dim3 dimBlock(N, N);
    MatAdd<<<1, dimBlock>>>(A, B, C);
}
```

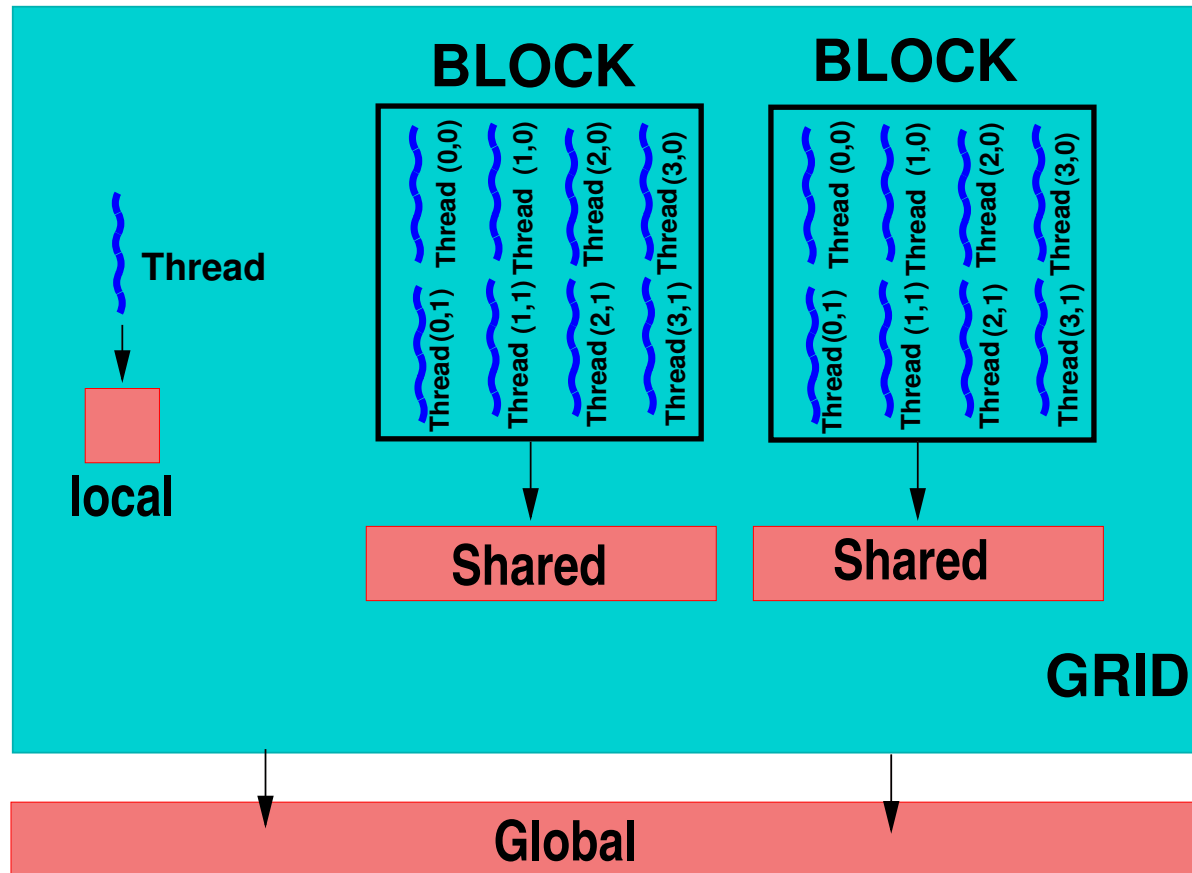
Example:

```
//device:  
__global__ void KernelFun(..)  
//host:  
dim3 DimGrid(200,10); //2000 thread blocks  
dim3 DimBlock(4,8,8); //256 threads per block  
size_t SharedMemBytes=64;//shared mem. per block  
KernelFun<<<DimGrid,DimBlock,SharedMemBytes>>>(..)
```

➤ How to get index of a thread?

- For a 1-D block: Index of a thread & its thread ID are the same
- For a 2-D block of size (D_x, D_y) : thread ID of a thread of index (x, y) is $(x + y * D_x)$;
- For 3-D blocks of size (D_x, D_y, D_z) : thread ID of a thread of index (x, y, z) is $(x + y * D_x + z * D_x * D_y)$.

CUDA environment: Memory Hierarchy



Threads can access their local memories, shared memory of their block, and global memory.

CUDA environment: Device & Host Memory

- Device (GPU) memory distinct from that of host.
- Kernels operate only on device memory
- Also: Texture memory [called CUDA arrays] –
- Can allocate device memory with `cudaMalloc()`
- Copy from host to device with `cudaMemcpy()`
- Can also use `cudaMallocPitch()`, `cudaMalloc3D()`, `cudaMemcpy2D()`, `cudaMemcpy3D()`, [see prog. guide]

CUDA environment: Shared vs. Global Memory

- By default, the kernel will use global memory
- However, shared memory is **much** faster and should be used when possible
- **Declarations:**

```
__shared__ float, int, ..
```

CUDA documentation, resources

➤ **Main documentation site:**

http://www.nvidia.com/object/cuda_develop.html

➤ **Short-cut: Latest programming guide for linux can be accessed from class site. See the 'programming' page.**

➤ **General information about CUDA**

www.nvidia.com/object/cuda_home

➤ **CUDA sample source code**

www.nvidia.com/object/cuda_get_samples

➤ **To download the CUDA SDK**

www.nvidia.com/object/cuda_get.html