Lecture 0: CUDA Introduction

Muhammad Osama Mahmoud, TA

Computer and Systems Eng. Dept. - Faculty of Engineering – University of Minia

Execution Model on Host and Device

- CUDA (Compute Unified Device Architecture) is a SPMD (Single Program Multiple Data)
- The serial parts of the code will run on CPU (Host)
- Host code is written in C/C++ programming language and executed on C
- And the parallel parts will run on GPU (Device)
- Device code is written in CUDA API

CUDA Parallel Threads

- A thread is a “virtualized” or “abstracted” Von-Neumann Processor
- CUDA is executed using Grid (array) of threads
- Since CUDA is SPMD, so all the threads runs the same program (code) in parallel on large amount of data
- Threads are identified by thread index

Data Parallelism "Vector Addition"

VectorA	$A[0]$	$A[1]$	$A[2]$	$A[3]$	…	$A[N-1]$	
	+	+	+	+	+	…	+
VectorB	$B[0]$	$B[1]$	$B[2]$	$B[3]$	…	$B[N-1]$	
	=	=	=	=	=	…	=
VectorC	$C[0]$	$C[1]$	$C[2]$	$C[3]$	…	$C[N-1]$	

CUDA Thread Blocks

- CUDA Thread array is divided into multiple Thread Blocks
- Threads within a block cooperate via Shared Memory, Atomic Operations and Barrier Synchronization
- Threads in different blocks do not interact
- Blocks are identified by block index

CUDA Built-in Varia

- **threadIdx**, 1D, 2D, 3D

An index to define the thread ID

- **blockIdx**, 1D, 2D, 3D

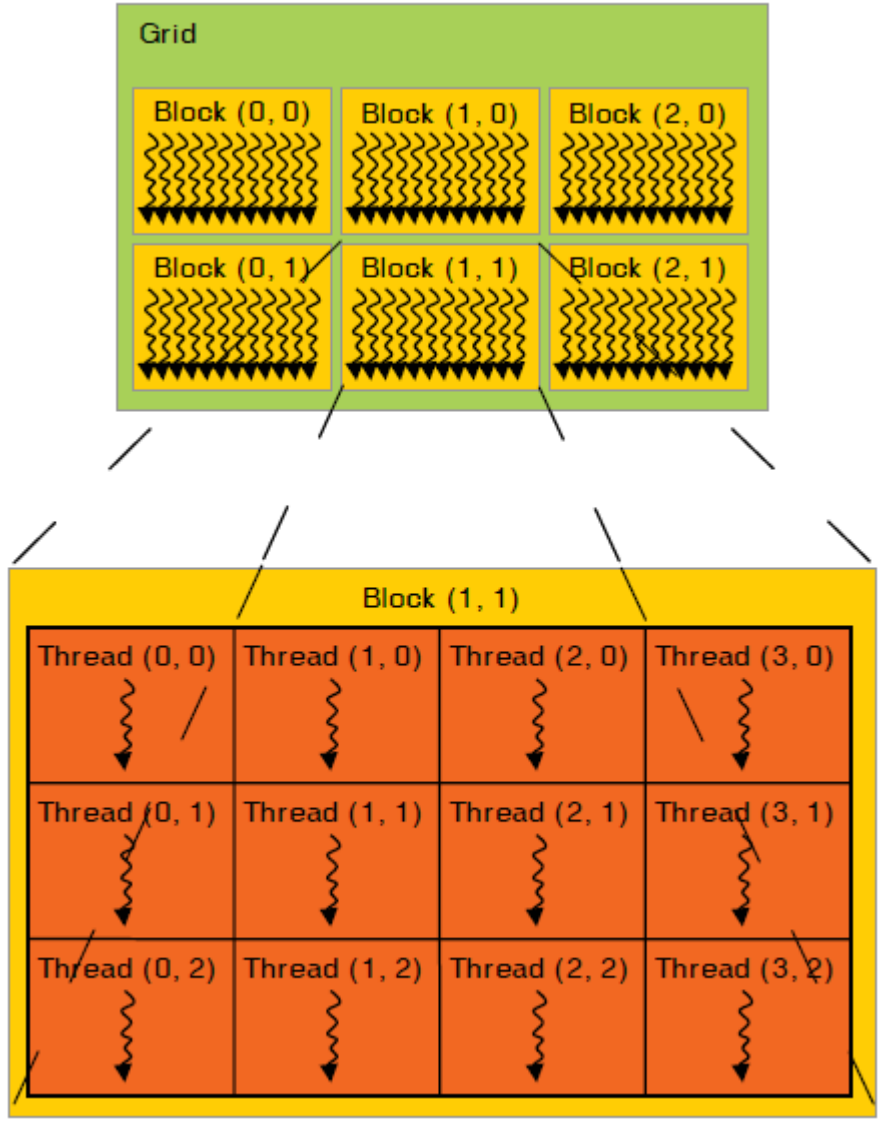
An index to define the block ID

- **blockDim**, 1D, 2D, 3D

Defines the dimensionality of each block

- **gridDim**, 1D, 2D, 3D

Defines the dimensionality of the grid



CUDA Function Declarations

Modifier	Executed on	Can be called from	Type
`__device__`	Device	Device	Device Function
`__global__`	Device	Host	Kernel `void` Function
`__host__`	Host	Host	Host Function

CUDA Function Declaration (Kernel)

```
__global__ void saxpy(int n, float a, float *x, float *y)
{
        int i = blockIdx.x*blockDim.x + threadIdx.x;
        if (i < n)
        {
                y[i] = a*x[i] + y[i];
        }
}
```

Kernel Launch (Host code)

```
int vecAdd(float* h_A, float* h_B, float* h_C, int n)
{
        // Allocate memory for GPU
        …
        // Run ceil(n/256.0) blocks of 256 threads each
        vecAddKernnel<<< ceil(n/256.0), 256 >>>(d_A, d_B, d_C, n);
}
```

Kernel Launch (Host code) (Cont.)

```
int vecAdd(float* h_A, float* h_B, float* h_C, int n)
{
        // Allocate memory for GPU
        …
        dim3 DimGrid((n-1)/256 + 1, 1, 1);
        dim3 DimBlock(256, 1, 1);
        vecAddKernnel<<< DimGrid, DimBlock >>>(d_A, d_B, d_C, n);
}
```

Acknowledgment

Thanks for Aiman Tarek for his help on these lecture notes

References

[1] Wen-mei W. Hwu, “Heterogeneous Parallel Programming”. Online course, 2014. Available: https://class.coursera.org/hetero-002

[2] NVIDIA, “CUDA C Programming Guide”, June 2014.