GPU Teaching Kit

Accelerated Computing
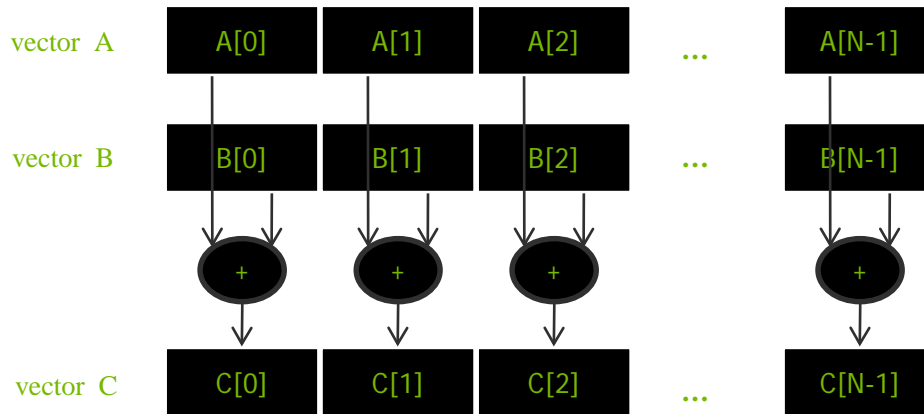
ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Lecture 2.3 – Introduction to CUDA C
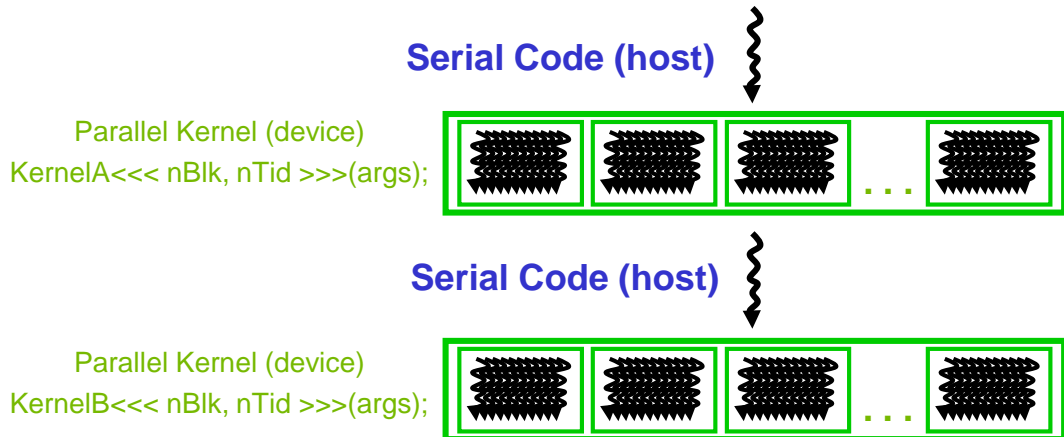
Threads and Kernel Functions

Objective

– To learn about CUDA threads, the main mechanism for exploiting of data parallelism
  – Hierarchical thread organization
  – Launching parallel execution
  – Thread index to data index mapping

2

Data Parallelism - Vector Addition Example

vector A — A[0] A[1] A[2] ... A[N-1]

vector B — B[0] B[1] B[2] ... B[N-1]

+ + + +
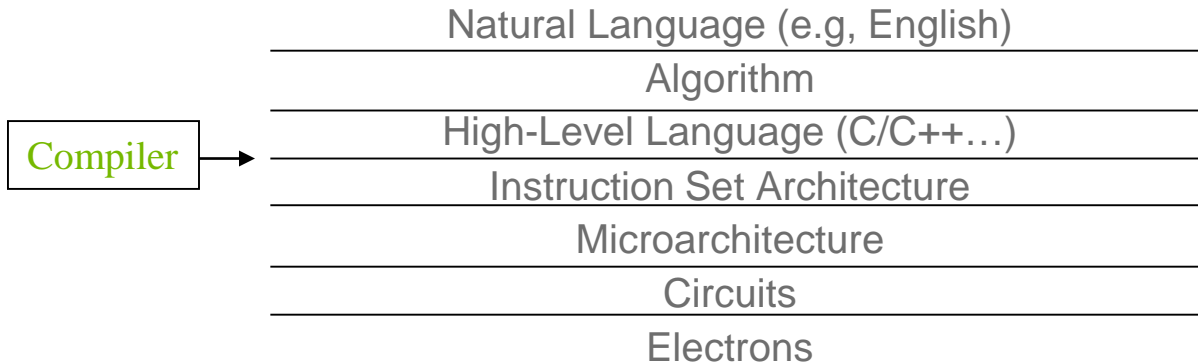
vector C — C[0] C[1] C[2] ... C[N-1]

CUDA Execution Model

– Heterogeneous host (CPU) + device (GPU) application C program
  – Serial parts in **host** C code
  – Parallel parts in **device** SPMD kernel code

Serial Code (host)

Parallel Kernel (device)
KernelA<<< nBlk, nTid >>>(args);

Serial Code (host)

Parallel Kernel (device)
KernelB<<< nBlk, nTid >>>(args);

From Natural Language to Electrons

Natural Language (e.g, English)

Algorithm

High-Level Language (C/C++…)

Compiler →

Instruction Set Architecture

Microarchitecture

Circuits

Electrons

©Yale Patt and Sanjay Patel, *From bits and bytes to gates and beyond*

NVIDIA   ILLINOIS

A program at the ISA level

– A program is a set of instructions stored in memory that can be read, interpreted, and executed by the hardware.

  – Both CPUs and GPUs are designed based on (different) instruction sets

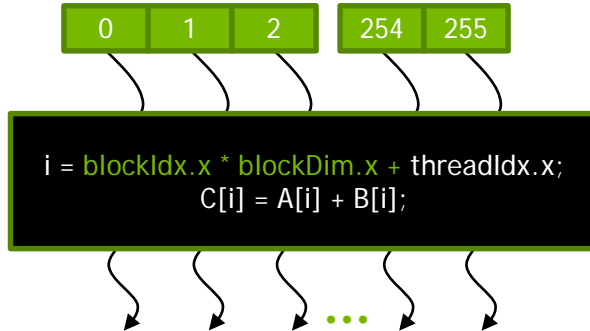– Program instructions operate on data stored in memory and/or registers.

A Thread as a Von-Neumann Processor

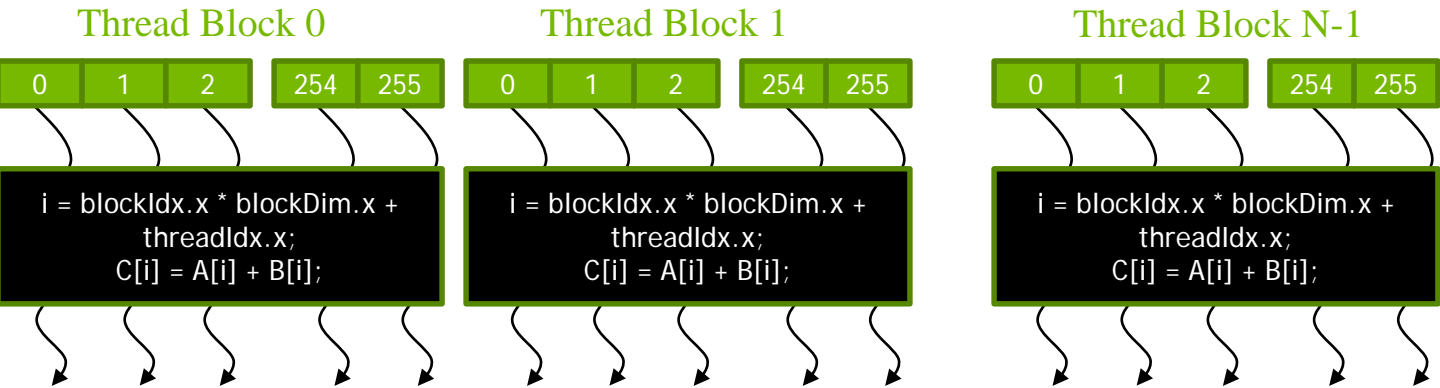A thread is a "virtualized" or "abstracted"
Von-Neumann Processor
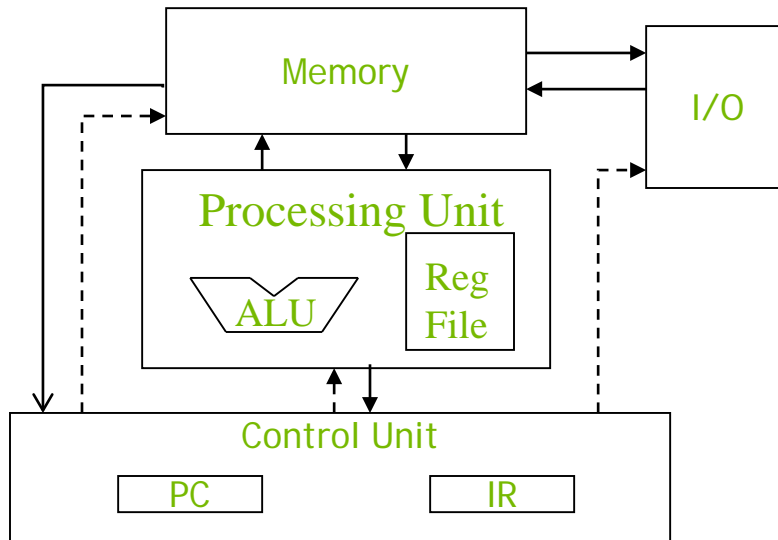
Arrays of Parallel Threads

- A CUDA kernel is executed by a grid (array) of threads
  - All threads in a grid run the same kernel code (Single Program Multiple Data)
  - Each thread has indexes that it uses to compute memory addresses and make control decisions
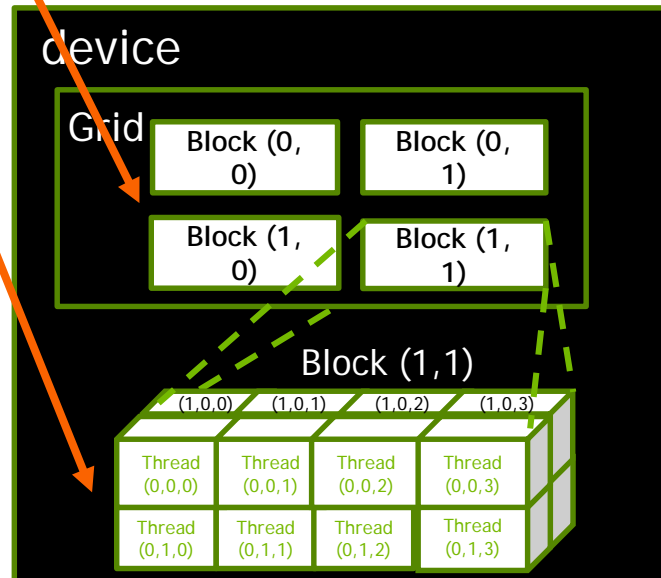
| 0 | 1 | 2 | | 254 | 255 |

```
i = blockIdx.x * blockDim.x + threadIdx.x;
            C[i] = A[i] + B[i];
```

. . .

Thread Blocks: Scalable Cooperation

Thread Block 0	Thread Block 1	Thread Block N-1

0	1	2		254	255

```
i = blockIdx.x * blockDim.x +
        threadIdx.x;
C[i] = A[i] + B[i];
```

0	1	2		254	255

```
i = blockIdx.x * blockDim.x +
        threadIdx.x;
C[i] = A[i] + B[i];
```

0	1	2		254	255

```
i = blockIdx.x * blockDim.x +
        threadIdx.x;
C[i] = A[i] + B[i];
```

– Divide thread array into multiple blocks
  – Threads within a block cooperate via **shared memory, atomic operations** and **barrier synchronization**
  – Threads in different blocks do not interact

blockIdx and threadIdx

- Each thread uses indices to decide what data to work on
  - blockIdx: 1D, 2D, or 3D (CUDA 4.0)
  - threadIdx: 1D, 2D, or 3D

- Simplifies memory addressing when processing multidimensional data
  - Image processing
  - Solving PDEs on volumes
  - …

device

Grid

Block (0, 0)   Block (0, 1)

Block (1, 0)   Block (1, 1)

Block (1,1)

(1,0,0) (1,0,1) (1,0,2) (1,0,3)

| Thread (0,0,0) | Thread (0,0,1) | Thread (0,0,2) | Thread (0,0,3) |
| Thread (0,1,0) | Thread (0,1,1) | Thread (0,1,2) | Thread (0,1,3) |

GPU Teaching Kit

Accelerated Computing