


Computer Structure and Language

Hamid Sarbazi-Azad
Department of Computer Engineering
Sharif University of Technology (SUT)
Tehran, Iran



1

(c) Hamid Sarbazi-Azad Computer Structure and Language – Lecture #29: GPU Introduction 2


A stylized, blue-toned illustration of an RTX 2080 Ti graphics card. The card is shown vertically, with three large fans visible on the left side. The text 'RTX 2080 Ti' is printed vertically on the right side of the card. The background is dark blue with some glowing lines and dots, suggesting a high-tech or digital environment.

Introduction to GPU & CUDA

2

(c) Hamid Sarbazi-Azad Computer Structure and Language – Lecture #29: GPU Introduction 3

GPU Architecture




3

(c) Hamid Sarbazi-Azad Computer Structure and Language – Lecture #29: GPU Introduction 4

Graphic Cards/ History

- 1980's – No GPUs. Just VGA controller
- 1990's – Add more function into VGA controller
- 1997 – 3D acceleration functions:
 - Hardware for triangle setup and rasterization
 - Texture mapping
 - Shading



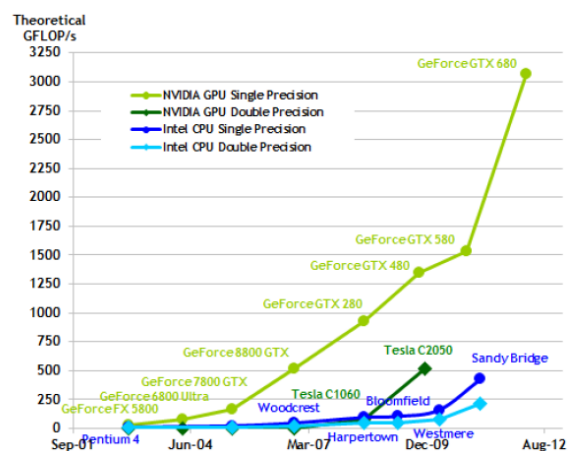
4

GPU History

- 2000 – A single chip graphics processor
 - beginning of GPU term
- 2005 – Massively parallel programmable processors
- 2007 – CUDA (Compute Unified Device Architecture)
 - Nvidia initiated
 - C/C++ extension
- 2008 – OpenCL
 - Apple initiated
 - Based on C99 standard

5

CPU vs. GPU




[CUDA_C_Programming Guide]

6

(c) Hamid Sarbazi-Azad Computer Structure and Language -- Lecture #29: GPU Introduction 7

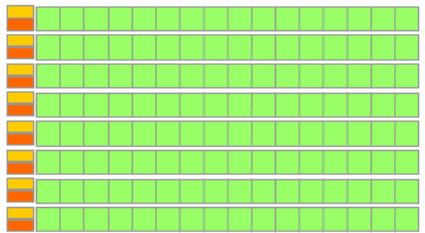
CPU vs. GPU (continued)

➤ CPU



The CPU diagram shows a yellow 'Control' block on the left, four green 'ALU' blocks arranged in a 2x2 grid to its right, and a large orange 'Cache' block below the ALUs.

➤ GPU



The GPU diagram shows eight horizontal rows, each representing a parallel processing unit. Each row starts with a small orange and yellow block, followed by a long green block.

[CUDA_C_Programming Guide]

7

(c) Hamid Sarbazi-Azad Computer Structure and Language -- Lecture #29: GPU Introduction 8

CPU vs. GPU (continued)

➤ CPU

- Latency oriented Cores
- Large caches
 - Lessen latency
- Sophisticated control
 - Branch prediction
 - Data forwarding
- Powerful ALUs
 - Reduce Latency

➤ GPU

- Throughput Oriented Cores
- Small caches
 - To boost memory throughput
- Simple control
 - No branch prediction
 - No data forwarding

8

(c) Hamid Sarbazi-Azad Computer Structure and Language -- Lecture #29: GPU Introduction 9

CPU vs. GPU (continued)

<ul style="list-style-type: none">➤ CPU<ul style="list-style-type: none">➤ Sequential parts where latency matters<ul style="list-style-type: none">➤ 10+X faster than GPU for sequential codes	<ul style="list-style-type: none">➤ GPU<ul style="list-style-type: none">➤ Heavily pipelined➤ Require massive number of threads to tolerate latencies
--	--

9

(c) Hamid Sarbazi-Azad Computer Structure and Language -- Lecture #29: GPU Introduction 10

Case Study

<ul style="list-style-type: none">➤ CPU<ul style="list-style-type: none">➤ Intel Core i7:960<ul style="list-style-type: none">➤ 4-core, 3.2 GHz➤ 2-way multi-threading➤ 4-way SIMD➤ L1 32KB, L2 256KB, L3 3MB➤ 32 GB/sec	<ul style="list-style-type: none">➤ GPU<ul style="list-style-type: none">➤ NVIDIA GTX 280<ul style="list-style-type: none">➤ 30 core, 1.3GHz➤ 1024-way multi-threading➤ 8-way SIMD➤ 16KB software managed cache (shared memory)➤ 141 GB/sec
--	---

10

(c) Hamid Sarbazi-Azad Computer Structure and Language -- Lecture #29: GPU Introduction 11

Number of Cores

- It is all about the core complexity:
 - The common goal: Improving pipeline efficiency
 - CPU goal: Single-thread performance
 - Exploiting ILP
 - Sophisticated branch predictor
 - Multiple issue logics
 - GPU goal: Throughput
 - Interleaving hundreds of threads

11

(c) Hamid Sarbazi-Azad Computer Structure and Language -- Lecture #29: GPU Introduction 12

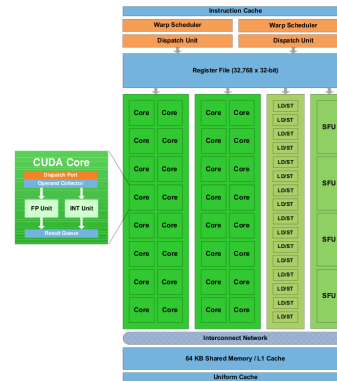
Cache Size

- CPU goal: reducing memory latency
 - Programmer-transparent data caching
 - Increasing the cache size to capture the working set
 - Prefetching (HW/SW)
- GPU goal: hiding memory latency
 - Interleave the execution of hundreds of threads to hide the latency of each other
- Notice:
 - CPU uses multi-threading for latency hiding
 - GPU uses software controlled caching (shared memory) for reducing memory latency

12

Streaming Multiprocessor (SM)

- Pipeline deep-multithreaded SIMD processor
- Multiple SIMD groups
- Resources shared among threads
 - Shared memory
 - Register file
 - L1Cache



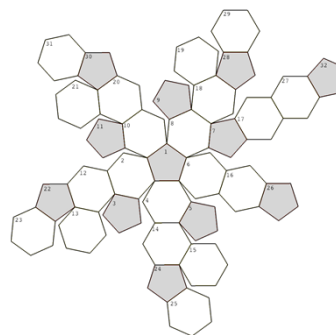
[NVIDIA Fermi Architecture Whitepaper]

13

Compute Capability (Nvidia)

- Denotes the Capability of the GPU

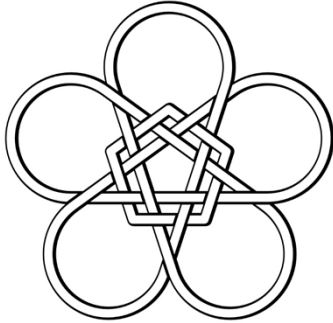
- Major . Minor
- 1.0 and 1.1
- 1.2
 - Atomic operation on shared memory
- 1.3
 - Double-precision operations
- 2.x and 3.x
 - Informative __syncthreads
 - 64-bit atomic
 - 3D grid
 - Tensor Cores
 - Mix Lib (Curand, CuFFT, ..)



14

(c) Hamid Sarbazi-Azad Computer Structure and Language – Lecture #29: GPU Introduction 15

GPU Programming Model



15

(c) Hamid Sarbazi-Azad Computer Structure and Language – Lecture #29: GPU Introduction 16

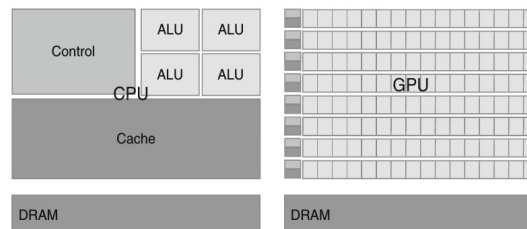
What Is CUDA

- Compute Unified Device Architecture
- Based on industry-standard C
- A handful of language extensions to allow heterogeneous programs
- Straightforward APIs to manage devices, memory, etc

16

Why CUDA?

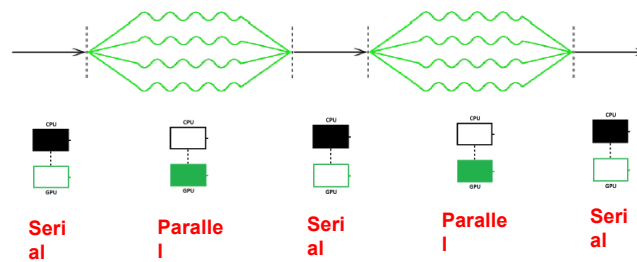
- Massive parallel computing power
- Maximize throughput of all threads
- Using hundreds of ALU inside a GPU



17

CUDA Programming Model

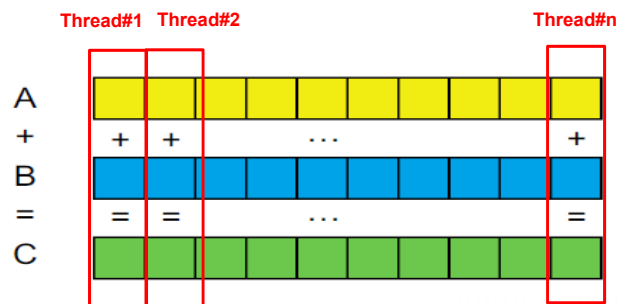
- Heterogeneous Programming
 - program separated into serial regions (run on CPU) & parallel regions (run on GPU)



18

CUDA Programming Model

- Parallel regions consist of many calculations that can be executed independently
 - Data Parallelism (e.g. vector addition)

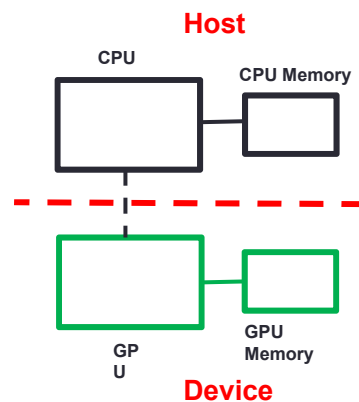


19

A Basic CUDA Program Outline

```

Int main() {
    // Allocate memory for array on host
    // Allocate memory for array on device
    // Fill array on host
    // copy data from host array to device array
    // Do something on device (e.g. vector addition)
    // Copy data from device array to host array
    // Check data for correctness
    // Free Host Memory
    // Free Device Memory }
  
```

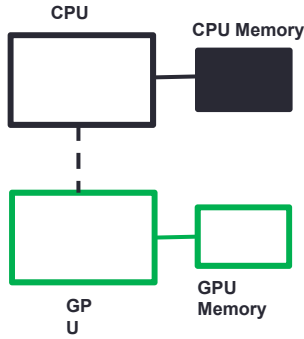


20

(c) Hamid Sarbazi-Azad Computer Structure and Language -- Lecture #29: GPU Introduction 21

A Basic CUDA Program Outline

```
Int main() {  
    // Allocate memory for array on host  
    // Allocate memory for array on device  
    // Fill array on host  
    // Copy data from host array to device array  
    // Do something on device (e.g. vector addition)  
    // Copy data from device array to host array  
    // Check data for correctness  
    // Free Host Memory  
    // Free Device Memory }  
}
```



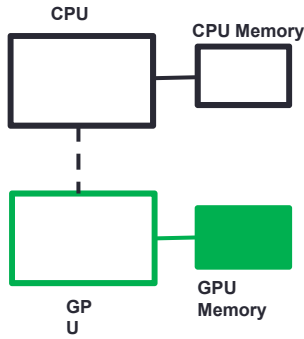
The diagram illustrates the hardware components for a GPU-based system. At the top, a white box labeled 'CPU' is connected to a dark grey box labeled 'CPU Memory'. Below the CPU, a dashed vertical line connects to a white box with a green border labeled 'GPU'. To the right of the GPU, a green box labeled 'GPU Memory' is connected to the GPU. The GPU and its memory are highlighted with green outlines.

21

(c) Hamid Sarbazi-Azad Computer Structure and Language -- Lecture #29: GPU Introduction 22

A Basic CUDA Program Outline

```
Int main() {  
    // Allocate memory for array on host  
    // Allocate memory for array on device  
    // Fill array on host  
    // Copy data from host array to device array  
    // Do something on device (e.g. vector addition)  
    // Copy data from device array to host array  
    // Check data for correctness  
    // Free Host Memory  
    // Free Device Memory }  
}
```



The diagram illustrates the hardware components for a GPU-based system. At the top, a white box labeled 'CPU' is connected to a white box labeled 'CPU Memory'. Below the CPU, a dashed vertical line connects to a white box with a green border labeled 'GPU'. To the right of the GPU, a green box labeled 'GPU Memory' is connected to the GPU. The GPU and its memory are highlighted with green outlines.

22

(c) Hamid Sarbazi-Azad Computer Structure and Language – Lecture #29: GPU Introduction 23

A Basic CUDA Program Outline

```

Int main() {

  // Allocate memory for array on host
  // Allocate memory for array on device

  // Fill array on host

  // Copy data from host array to device array
  // Do something on device (e.g. vector addition)
  // Copy data from device array to host array

  // Check data for correctness

  // Free Host Memory
  // Free Device Memory }

```

The diagram illustrates the initial state of a GPU program. It shows a CPU (black box) connected to CPU Memory (white box). A GPU (green box) is connected to GPU Memory (green box). A dashed line connects the CPU and GPU, indicating communication.

23

(c) Hamid Sarbazi-Azad Computer Structure and Language – Lecture #29: GPU Introduction 24

A Basic CUDA Program Outline

```

Int main() {

  // Allocate memory for array on host
  // Allocate memory for array on device

  // Fill array on host

  // Copy data from host array to device array
  // Do something on device (e.g. vector addition)
  // Copy data from device array to host array

  // Check data for correctness

  // Free Host Memory
  // Free Device Memory }

```

The diagram illustrates the state of a GPU program after data transfer. It shows a CPU (white box) connected to CPU Memory (black box). The GPU (green box) is connected to GPU Memory (green box). A dashed line connects the CPU and GPU. A large white arrow points from CPU Memory to GPU Memory, indicating data transfer.

24

(c) Hamid Sarbazi-Azad Computer Structure and Language -- Lecture #29: GPU Introduction 25

A Basic CUDA Program Outline

```

Int main() {

// Allocate memory for array on host
// Allocate memory for array on device
// Fill array on host
// Copy data from host array to device array
// Do something on device (e.g. vector addition)
// Copy data from device array to host array
// Check data for correctness
// Free Host Memory
// Free Device Memory }
  
```

The diagram illustrates the hardware components for a CUDA program. At the top, a white box labeled 'CPU' is connected to a white box labeled 'CPU Memory'. Below the CPU, a dashed vertical line connects to a green box labeled 'GPU'. The GPU is connected to a green box labeled 'GPU Memory'.

25

(c) Hamid Sarbazi-Azad Computer Structure and Language -- Lecture #29: GPU Introduction 26

A Basic CUDA Program Outline

```

Int main() {

// Allocate memory for array on host
// Allocate memory for array on device
// Fill array on host
// Copy data from host array to device array
// Do something on device (e.g. vector addition)
// Copy data from device array to host array
// Check data for correctness
// Free Host Memory
// Free Device Memory }
  
```

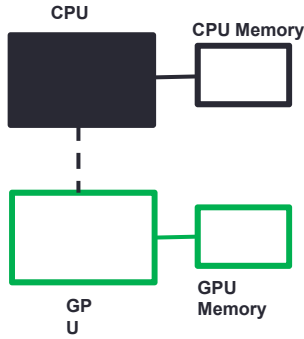
The diagram illustrates the hardware components for a CUDA program, similar to the previous slide. At the top, a white box labeled 'CPU' is connected to a white box labeled 'CPU Memory'. Below the CPU, a dashed vertical line connects to a green box labeled 'GPU'. The GPU is connected to a green box labeled 'GPU Memory'. A white arrow points from the 'GPU Memory' box up to the 'CPU Memory' box, indicating data transfer from the device to the host.

26

(c) Hamid Sarbazi-Azad Computer Structure and Language -- Lecture #29: GPU Introduction 27

A Basic CUDA Program Outline

```
Int main() {  
    // Allocate memory for array on host  
    // Allocate memory for array on device  
    // Fill array on host  
    // Copy data from host array to device array  
    // Do something on device (e.g. vector addition)  
    // Copy data from device array to host array  
    // Check data for correctness  
    // Free Host Memory  
    // Free Device Memory }  
}
```



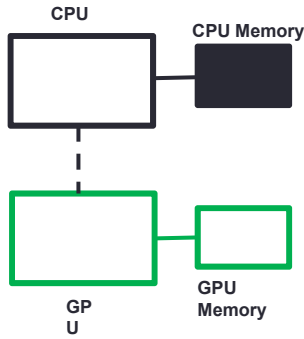
The diagram illustrates the hardware components and their connections. At the top, a black box labeled 'CPU' is connected to a white box labeled 'CPU Memory' by a solid line. Below the CPU, a dashed vertical line connects to a green-outlined box labeled 'GPU'. To the right of the GPU, a green-outlined box labeled 'GPU Memory' is connected to the GPU by a solid line.

27

(c) Hamid Sarbazi-Azad Computer Structure and Language -- Lecture #29: GPU Introduction 28

A Basic CUDA Program Outline

```
Int main() {  
    // Allocate memory for array on host  
    // Allocate memory for array on device  
    // Fill array on host  
    // Copy data from host array to device array  
    // Do something on device (e.g. vector addition)  
    // Copy data from device array to host array  
    // Check data for correctness  
    // Free Host Memory  
    // Free Device Memory }  
}
```



The diagram illustrates the hardware components and their connections. At the top, a white box labeled 'CPU' is connected to a black box labeled 'CPU Memory' by a solid line. Below the CPU, a dashed vertical line connects to a green-outlined box labeled 'GPU'. To the right of the GPU, a green-outlined box labeled 'GPU Memory' is connected to the GPU by a solid line.

28

(c) Hamid Sarbazi-Azad Computer Structure and Language -- Lecture #29: GPU Introduction 29

A Basic CUDA Program Outline

```

Int main() {
    // Allocate memory for array on host
    // Allocate memory for array on device
    // Fill array on host
    // Copy data from host array to device array
    // Do something on device (e.g. vector addition)
    // Copy data from device array to host array
    // Check data for correctness
    // Free Host Memory
    // Free Device Memory }

```

The diagram illustrates the hardware components: a CPU box connected to a CPU Memory box, and a GPU box connected to a GPU Memory box. A vertical dashed line connects the CPU and GPU boxes, representing the system bus or communication channel.

29

(c) Hamid Sarbazi-Azad Computer Structure and Language -- Lecture #29: GPU Introduction 30

A Basic CUDA Program Outline

```

Int main() {
    // Allocate memory for array on host
    size_t bytes = N*sizeof(int);
    int *A = (int*)malloc(bytes);
    int *B = (int*)malloc(bytes);
    int *C = (int*)malloc(bytes);
    ...
}

```

The diagram illustrates the hardware components: a CPU box connected to a CPU Memory box, and a GPU box connected to a GPU Memory box. A vertical dashed line connects the CPU and GPU boxes, representing the system bus or communication channel.

30

(c) Hamid Sarbazi-Azad Computer Structure and Language -- Lecture #29: GPU Introduction 31

A Basic CUDA Program Outline

```

Int main() {
    ...

    // Allocate memory for array on device

    int *d_A, *d_B, *d_C;

    cudaMalloc(&d_A, bytes);

    cudaMalloc(&d_B, bytes);

    cudaMalloc(&d_C, bytes);

    ...
}

```

The diagram illustrates the hardware components for a CUDA program. It shows a CPU box connected to a CPU Memory box. Below them, a GPU box is connected to a GPU Memory box. A vertical dashed line connects the CPU and GPU boxes, representing the system bus or communication channel.

31

(c) Hamid Sarbazi-Azad Computer Structure and Language -- Lecture #29: GPU Introduction 32

A Basic CUDA Program Outline

```

Int main() {
    ...

    // Fill array on host

    for(int i=0; i<N; i++)
    {
        A[i] = 1;
        B[i] = 2;
        C[i] = 0;
    }

    ...
}

```

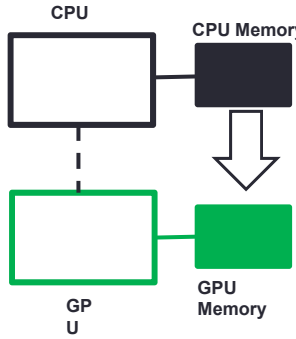
The diagram illustrates the hardware components for a CUDA program, identical to the previous slide. It shows a CPU box connected to a CPU Memory box. Below them, a GPU box is connected to a GPU Memory box. A vertical dashed line connects the CPU and GPU boxes, representing the system bus or communication channel.

32

(c) Hamid Sarbazi-Azad Computer Structure and Language -- Lecture #29: GPU Introduction 33

A Basic CUDA Program Outline

```
Int main() {  
    ...  
    // Copy data from host array to device array  
    cudaMemcpy(d_A, A, bytes,  
               cudaMemcpyHostToDevice);  
    cudaMemcpy(d_B, B, bytes,  
               cudaMemcpyHostToDevice);  
    ...  
}
```



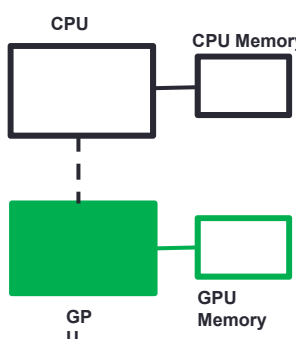
The diagram illustrates the hardware components and data flow. At the top, a white box labeled 'CPU' is connected to a dark grey box labeled 'CPU Memory'. Below the CPU, a dashed vertical line connects to a green-outlined box labeled 'GPU'. To the right of the GPU is a green box labeled 'GPU Memory', connected to the GPU by a solid green line. A large white arrow points from the 'CPU Memory' box down to the 'GPU Memory' box, indicating the transfer of data from host memory to device memory.

33

(c) Hamid Sarbazi-Azad Computer Structure and Language -- Lecture #29: GPU Introduction 34

A Basic CUDA Program Outline

```
Int main() {  
    ...  
    // Do something on device (e.g. vector addition)  
    // We'll come back to this soon  
    ...  
}
```



The diagram illustrates the hardware components and data flow. At the top, a white box labeled 'CPU' is connected to a white box labeled 'CPU Memory'. Below the CPU, a dashed vertical line connects to a green-outlined box labeled 'GPU'. To the right of the GPU is a white box labeled 'GPU Memory', connected to the GPU by a solid green line.

34

(c) Hamid Sarbazi-Azad Computer Structure and Language -- Lecture #29: GPU Introduction 35

A Basic CUDA Program Outline

```

Int main() {
    ...

    // Copy data from device array to host array
    cudaMemcpy(C, d_C, bytes,
               cudaMemcpyDeviceToHost);

    ...
}

```

The diagram illustrates the hardware components and data flow. At the top, a white box labeled 'CPU' is connected to a dark grey box labeled 'CPU Memory'. At the bottom, a green-outlined box labeled 'GPU' is connected to a green box labeled 'GPU Memory'. A dashed vertical line connects the CPU and GPU boxes. A white arrow points upwards from the GPU Memory box to the CPU Memory box, indicating data transfer from the device to the host.

35

(c) Hamid Sarbazi-Azad Computer Structure and Language -- Lecture #29: GPU Introduction 36

A Basic CUDA Program Outline

```

Int main() {
    ...

    // Check data for correctness
    for (int i=0; i<N; i++)
    {
        if(C[i] != 3)
        {
            // Error – value of C[i] is not correct!
        }
    }

    ...
}

```

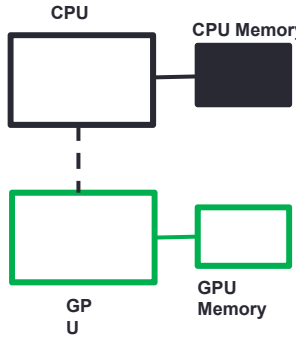
The diagram illustrates the hardware components and data flow. At the top, a black box labeled 'CPU' is connected to a white box labeled 'CPU Memory'. At the bottom, a green-outlined box labeled 'GPU' is connected to a green box labeled 'GPU Memory'. A dashed vertical line connects the CPU and GPU boxes. A white arrow points downwards from the CPU Memory box to the GPU Memory box, indicating data transfer from the host to the device.

36

(c) Hamid Sarbazi-Azad Computer Structure and Language -- Lecture #29: GPU Introduction 37

A Basic CUDA Program Outline

```
Int main() {  
    ...  
    // Free Host Memory  
    free(A);  
    free(B);  
    free(C);  
    ...  
}
```



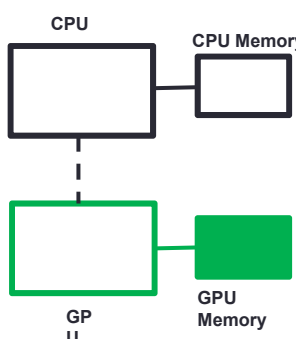
The diagram illustrates the hardware components of a system. At the top, a box labeled 'CPU' is connected to a dark grey box labeled 'CPU Memory'. Below the CPU, a dashed vertical line connects to a green-outlined box labeled 'GPU'. To the right of the GPU, a green-outlined box labeled 'GPU Memory' is connected to the GPU by a solid green line.

37

(c) Hamid Sarbazi-Azad Computer Structure and Language -- Lecture #29: GPU Introduction 38

A Basic CUDA Program Outline

```
Int main() {  
    ...  
    // Free Device Memory  
    cudaFree(d_A);  
    cudaFree(d_B);  
    cudaFree(d_C);  
    ...  
}
```



The diagram illustrates the hardware components of a system. At the top, a box labeled 'CPU' is connected to a white box labeled 'CPU Memory'. Below the CPU, a dashed vertical line connects to a green-outlined box labeled 'GPU'. To the right of the GPU, a solid green box labeled 'GPU Memory' is connected to the GPU by a solid green line.

38

(c) Hamid Sarbazi-Azad Computer Structure and Language – Lecture #29: GPU Introduction 39

CUDA Kernels

- What is difference between serial and parallel implementation?

Serial – CPU

```
for (int i=0; i<N; i++){
    C[i] = A[i] + B[i];
}
```

Parallel – GPU

```
C[i] = A[i] + B[i];
```

- A kernel is a function executed on the GPU as an array of threads in parallel
- Same code is executed by all threads
 - Single-Program Multiple-Data (SPMD)
- Each thread has:
 - thread ID
 - inputs, and output results

39

(c) Hamid Sarbazi-Azad Computer Structure and Language – Lecture #29: GPU Introduction 40

CUDA Kernels

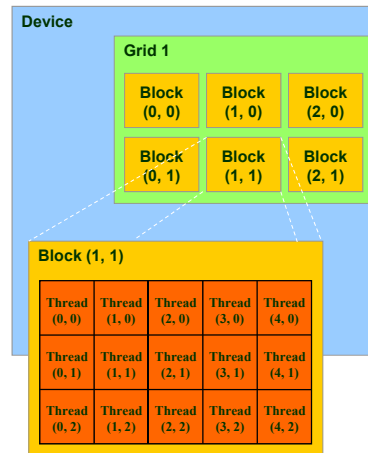
- Threads are grouped into blocks
- Blocks are grouped into a grid

Kernel is executed as a grid of blocks of threads

40

Built-in Variables

- **gridDim**: Grid dimension
- **blockDim**: Block dimension
- **blockIdx**: Block index
- **threadIdx**: Thread index



41

Execution Configuration

- 1D grid / 1D blocks

gridDim.x = 1024 **blockDim.x = 64**
gridDim.y = 1 **blockDim.y = 1**
blockDim.z = 1

```
dim3 gd(1024)
dim3 bd(64)
akernel<<<gd, bd>>>(...)
```

- 2D grid / 3D blocks

gridDim.x = 4 **blockDim.x = 64**
gridDim.y = 128 **blockDim.y = 16**
blockDim.z = 4

```
dim3 gd(4, 128)
dim3 bd(64, 16, 4)
akernel<<<gd, bd>>>(...)
```

42

(c) Hamid Sarbazi-Azad

Computer Structure and Language -- Lecture #29: GPU Introduction

43

CUDA Function Declarations

Functions	Executed on the:	Only callable from the:
<code>__device__ float DeviceFunc()</code>	device	device
<code>__global__ void KernelFunc()</code>	device	host
<code>__host__ float HostFunc()</code>	host	host

- `__global__` defines a kernel function
 - Must return void
 - Can only call `__device__` functions

43

(c) Hamid Sarbazi-Azad

Computer Structure and Language -- Lecture #29: GPU Introduction

44

Vector Addition Kernel (1)

- Device code

```
__global__ vector_addition(int *a, int *b, int *c)
{
    int i = blockIdx.x;
    c[i] = a[i] + b[i];
}
```
- Host code

```
Int main(){
...
vector_addition <<N, 1>>> (d_A, d_B, d_C);
}
```

N blocks

Each block has one thread

44

(c) Hamid Sarbazi-Azad Computer Structure and Language -- Lecture #29: GPU Introduction 45

Vector Addition Kernel (2)

- Device code

```
__global__ vector_addition (int *a, int *b, int *c)
{
    int i = threadIdx.x;
    c[i] = a[i] + b[i];
}
```

What is wrong with this code?

- Host code

```
Int main(){
    ...
    vector_addition <<< 1, N >>> (d_A, d_B, d_C);
}
```

One block with N threads

45

(c) Hamid Sarbazi-Azad Computer Structure and Language -- Lecture #29: GPU Introduction 46

Vector Addition Kernel (3)

- Device code

```
__global__ vector_addition (int *a, int *b, int *c)
{
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    if (i < N)
        c[i] = a[i] + b[i];
}
```

Blocks

46

(c) Hamid Sarbazi-Azad Computer Structure and Language – Lecture #29: GPU Introduction 47

Vector Addition Kernel (3)

- Device code

```
__global__ vector_addition (int *a, int *b, int *c)
{
    int i = blockDim.x * blockIdx.x + threadIdx.x;

    if (i < N)
        c[i] = a[i] + b[i];
}
```

Threads

47

(c) Hamid Sarbazi-Azad Computer Structure and Language – Lecture #29: GPU Introduction 48

Vector Addition Kernel (3)

- Device code

```
__global__ vector_addition (int *a, int *b, int *c)
{
    int i = (4) * (0-3) + (0-3);

    if (i < N)
        c[i] = a[i] + b[i];
}
```

0

1

2

3

48

(c) Hamid Sarbazi-Azad
Computer Structure and Language -- Lecture #29: GPU Introduction
49

Vector Addition Kernel (3)

- Device code

```

__global__ vector_addition (int *a, int *b, int *c)
{
    (4)      (2)      (1)
    int i = blockDim.x * blockIdx.x + threadIdx.x;

    if (i < N)
        c[i] = a[i] + b[i];
}

```

0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3

block 0 block 1 block 2 block 3

int i = 4 * 2 + 1 = 9

A₀ A₁ A₂ A₃ A₄ A₅ A₆ A₇ A₈ A₉ A₁₀ A₁₁ A₁₂ A₁₃ A₁₄ A₁₅

49

(c) Hamid Sarbazi-Azad

Computer Structure and Language -- Lecture #29: GPU Introduction

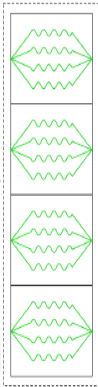
50

Vector Addition Kernel (3)

- Device code

```
__global__ vector_addition (int *a, int *b, int *c)
{
    int i = blockDim.x * blockIdx.x + threadIdx.x;

    if (i < N)
        c[i] = a[i] + b[i];
}
```



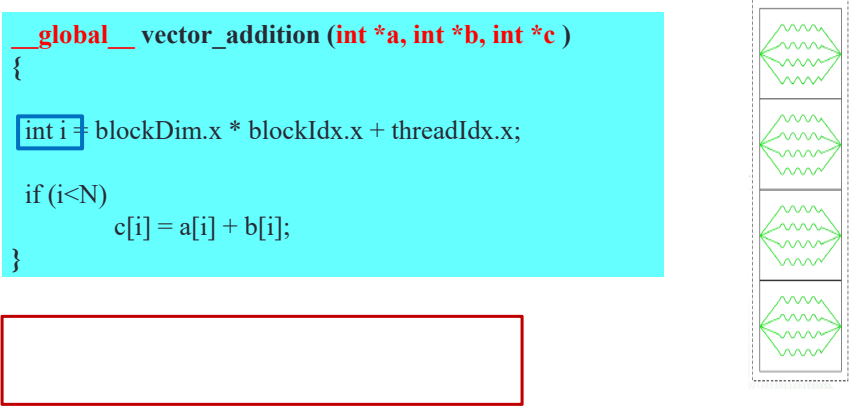
50

(c) Hamid Sarbazi-Azad Computer Structure and Language – Lecture #29: GPU Introduction 51

Vector Addition Kernel (3)

- Device code

```
__global__ vector_addition (int *a, int *b, int *c )  
{  
    int i = blockDim.x * blockIdx.x + threadIdx.x;  
    if (i < N)  
        c[i] = a[i] + b[i];  
}
```

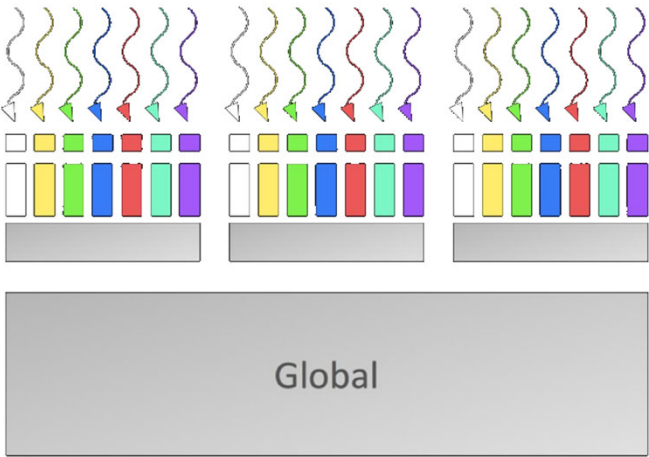


The diagram illustrates a GPU architecture. On the right, a vertical stack of four green wavy lines represents threads. Below them, a red rectangle represents a memory block. The threads are connected to the memory block, indicating data flow.

51

(c) Hamid Sarbazi-Azad Computer Structure and Language – Lecture #29: GPU Introduction 52

Memory Model



The diagram illustrates a GPU memory model. It shows three columns of threads (wavy lines) connected to a Global memory block. Each column has a set of threads (wavy lines) and a set of registers (colored boxes). The threads are connected to the registers, and the registers are connected to the Global memory block. The Global memory block is labeled "Global".

52

(c) Hamid Sarbazi-Azad Computer Structure and Language -- Lecture #29: GPU Introduction 53

Shared Memory

- Very fast on-chip memory
- Allocated per thread block
 - Allows data sharing between threads in the same block
 - Declared with `__shared__` specifier
- Limited amount
- Must take care to avoid race conditions. For example...
 - Say, each thread writes the value 1 to one element of an array element
 - Then one thread sums up the elements of the array
 - Synchronize with `__syncthreads()`

53

(c) Hamid Sarbazi-Azad Computer Structure and Language -- Lecture #29: GPU Introduction 54

SM EXECUTION & DIVERGENCE

54

(c) Hamid Sarbazi-Azad Computer Structure and Language – Lecture #29: GPU Introduction 55

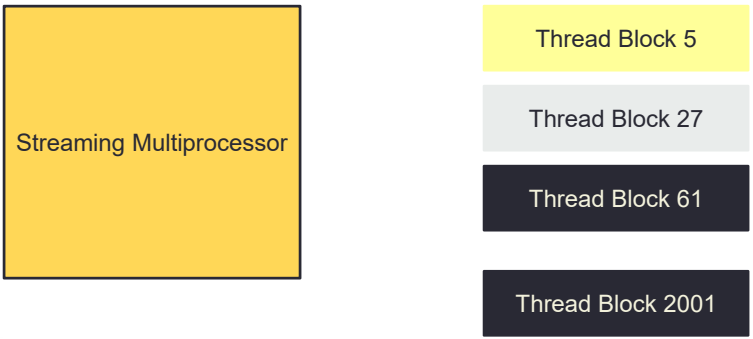
How an SM executes threads

- Overview of how a Stream Multiprocessor works
- SIMT Execution
- Divergence

55

(c) Hamid Sarbazi-Azad Computer Structure and Language – Lecture #29: GPU Introduction 56

Scheduling Blocks onto SMs



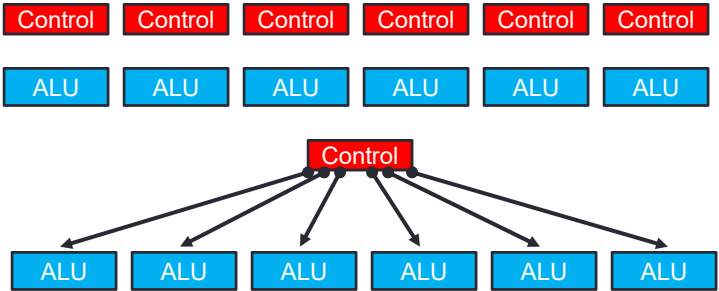
The diagram illustrates the scheduling of thread blocks onto a Streaming Multiprocessor (SM). On the left, a yellow square represents the 'Streaming Multiprocessor'. To its right, a vertical queue of four thread blocks is shown: 'Thread Block 5' (yellow), 'Thread Block 27' (light gray), 'Thread Block 61' (dark gray), and 'Thread Block 2001' (dark gray). The blocks are scheduled in order of their completion, with the first block in the queue being the one currently executing on the SM.

- **HW Schedules thread blocks onto available SMs**
 - No guarantee of ordering among thread blocks
 - HW will schedule thread blocks as soon as a previous thread block finishes

56

(c) Hamid Sarbazi-Azad Computer Structure and Language – Lecture #29: GPU Introduction 57

Warps



- A **warp** = 32 threads launched together
- Usually, execute together as well

57

(c) Hamid Sarbazi-Azad Computer Structure and Language – Lecture #29: GPU Introduction 58

Control Flow Divergence

- What happens if you have the following code?

```
if (foo (threadIdx.x) )
{
    do_A() ;
}
else
{
    do_B() ;
}
```

58

(c) Hamid Sarbazi-Azad Computer Structure and Language – Lecture #29: GPU Introduction 59

Control Flow Divergence

From Fung et al. MICRO '07

59

(c) Hamid Sarbazi-Azad Computer Structure and Language – Lecture #29: GPU Introduction 60

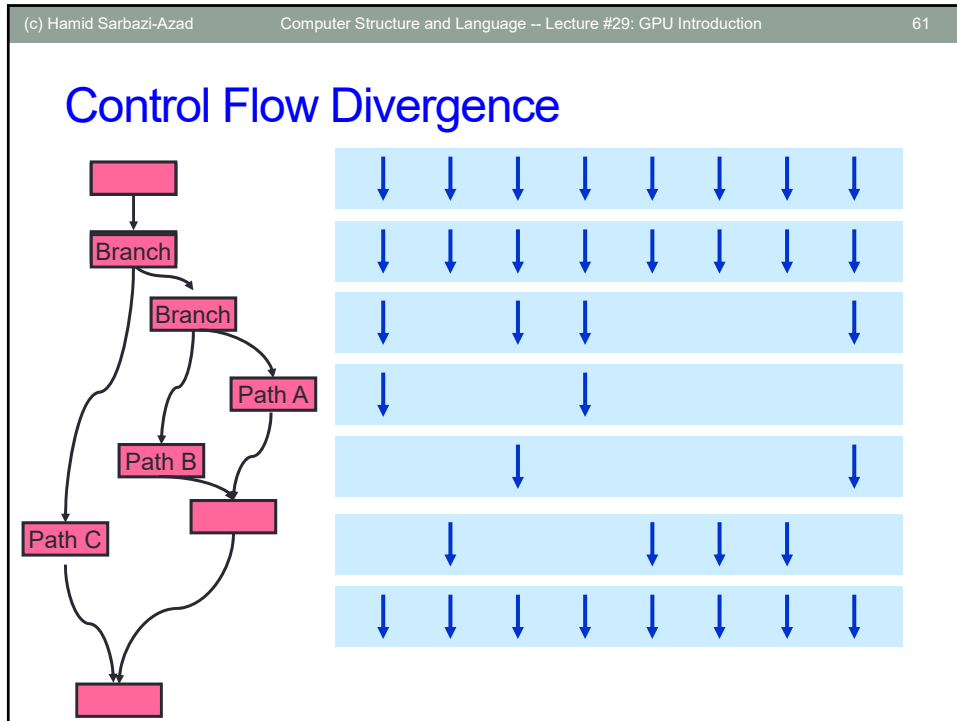
Control Flow Divergence

- Nested branches are handled as well

```

if (foo(threadIdx.x))
{
    if (bar(threadIdx.x))
        do_A();
    else
        do_B();
}
else
    do_C();
  
```

60



(c) Hamid Sarbazi-Azad Computer Structure and Language -- Lecture #29: GPU Introduction 62

Control Flow Divergence

- You don't have to worry about divergence for correctness (*)
- You might have to think about it for performance
 - Depends on your branch conditions

62

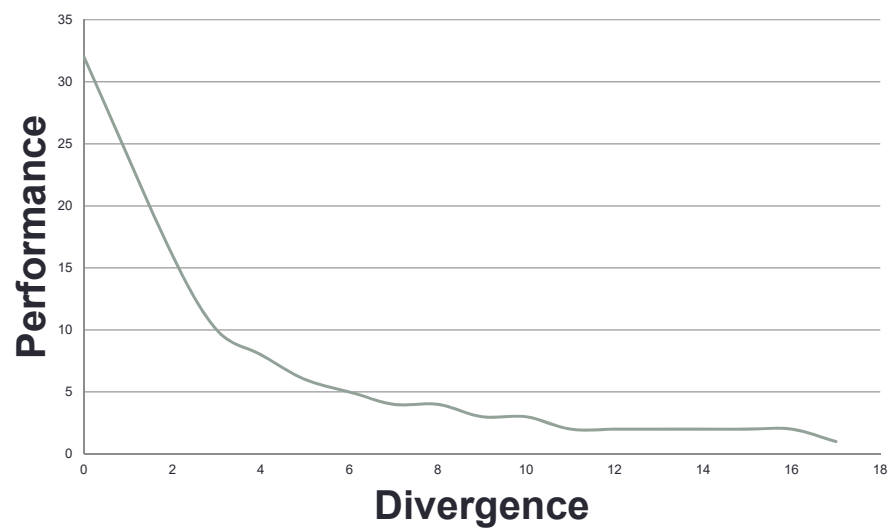
Control Flow Divergence

- Performance drops off with the degree of divergence

```
switch(threadIdx.x % N)
{
    case 0:
        ...
    case 1:
        ...
}
```

63

Divergence



64

END OF SLIDES

65