GPU Memory Hierarchy

Outline

- To learn to efficiently use the memory hierarchy inside a parallel program

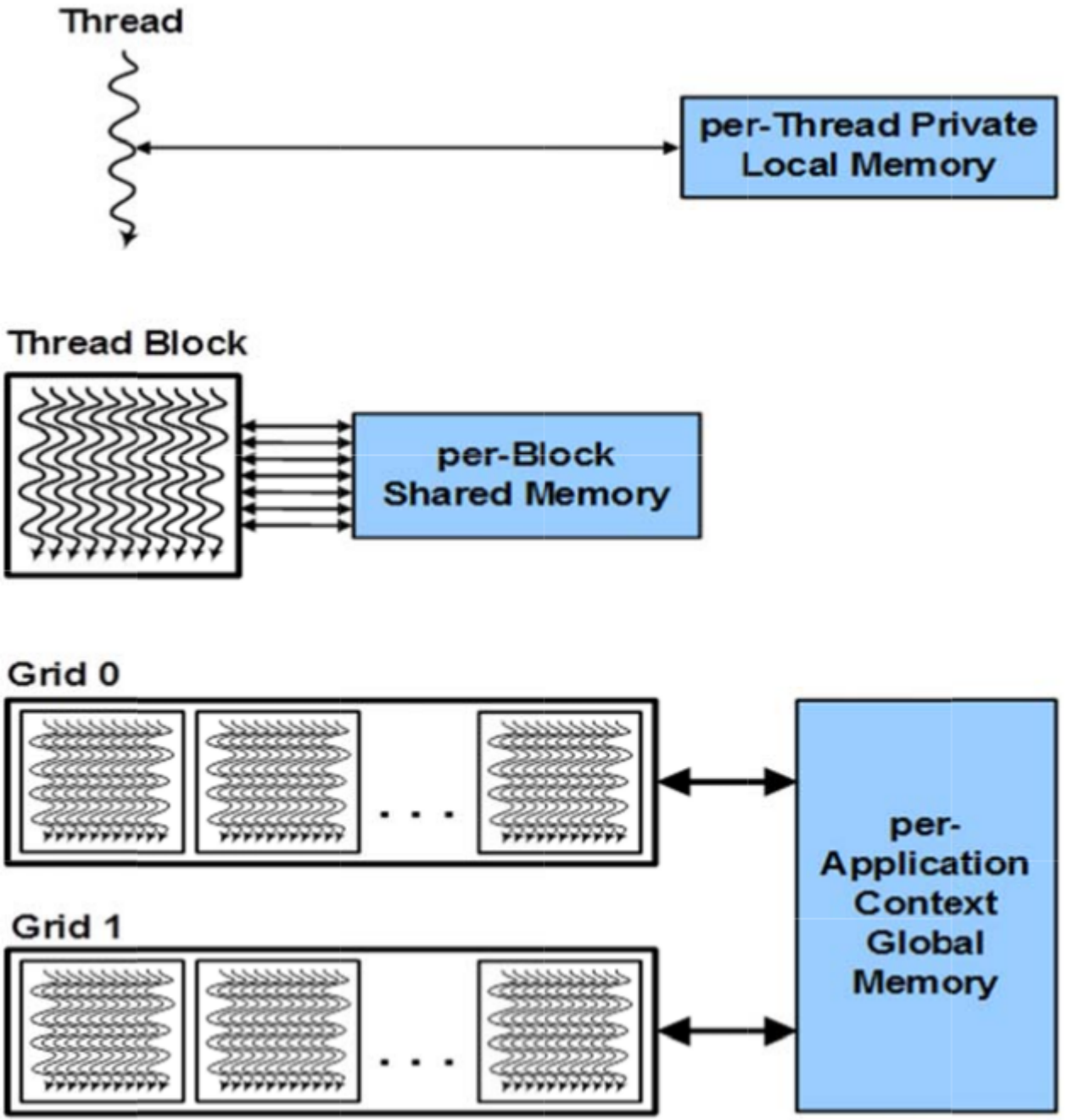
Outline

- To learn to efficiently use the memory hierarchy inside a parallel program
- Registers, shared memory, global memory

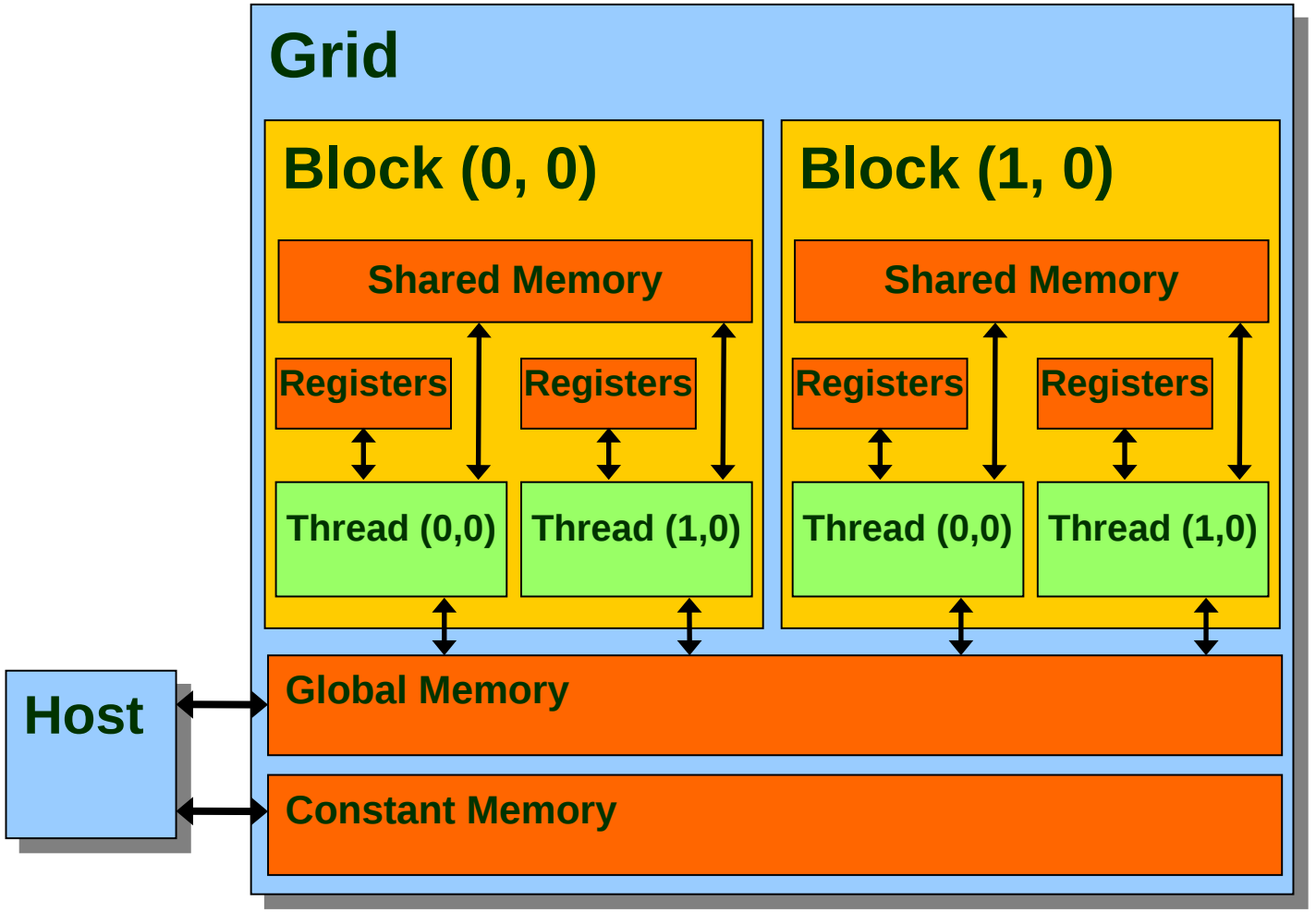
Outline

- To learn to efficiently use the memory hierarchy inside a parallel program
- Registers, shared memory, global memory
- To magnify memory bandwidth for parallel execution

Thread & Memory Hierarchy



Programmer View of CUDA Memories



CUDA Variables

Variable declaration	Memory	Scope	Lifetime
`int LocalVar;`	register	thread	thread

CUDA Variables

Variable declaration	Memory	Scope	Lifetime
`int LocalVar;`	register	thread	thread
`__device__ __shared__` `int SharedVar;`	shared	block	block

CUDA Variables

Variable declaration	Memory	Scope	Lifetime
`int LocalVar;`	register	thread	thread
`__device__ __shared__` `int SharedVar;`	shared	block	block
`__device__` `int GlobalVar;`	global	grid	application

CUDA Variables

Variable declaration	Memory	Scope	Lifetime
`int LocalVar;`	register	thread	thread
`__device__ __shared__` `int SharedVar;`	shared	block	block
`__device__` `int GlobalVar;`	global	grid	application
`__device__ __constant__` `int ConstantVar;`	constant	grid	application

CUDA Variables

Variable declaration	Memory	Scope	Lifetime
`int LocalVar;`	register	thread	thread
`__device__ __shared__` `int SharedVar;`	shared	block	block
`__device__` `int GlobalVar;`	global	grid	application
`__device__ __constant__` `int ConstantVar;`	constant	grid	application

`__device__` is optional when used with **`__shared__`**, or **`__constant__`**

CUDA Variables

Variable declaration	Memory	Scope	Lifetime
`int LocalVar;`	register	thread	thread
`__device__ __shared__` `int SharedVar;`	shared	block	block
`__device__` `int GlobalVar;`	global	grid	application
`__device__ __constant__` `int ConstantVar;`	constant	grid	application

`__device__` is optional when used with **`__shared__`**, or **`__constant__`**

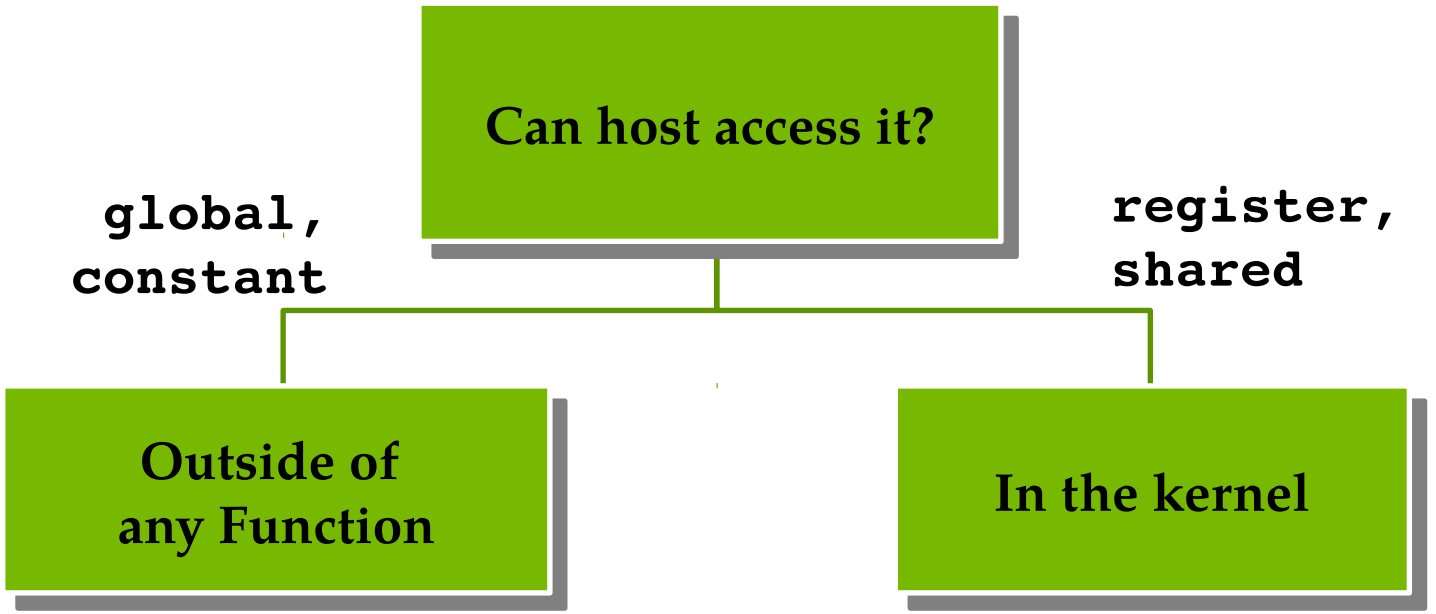
Automatic variables reside in a **register**

CUDA Variables

Variable declaration	Memory	Scope	Lifetime
`int LocalVar;`	register	thread	thread
`__device__ __shared__` `int SharedVar;`	shared	block	block
`__device__` `int GlobalVar;`	global	grid	application
`__device__ __constant__` `int ConstantVar;`	constant	grid	application

- **`__device__`** is optional when used with **`__shared__`**, or **`__constant__`**
- Automatic variables reside in a **register**
  - Except per-thread arrays that reside in global memory

Where to declare variables?



Shared Memory in CUDA

- Is explicitly defined and used in the kernel code

Shared Memory in CUDA

- Is explicitly defined and used in the kernel code
- One in each SM

Shared Memory in CUDA

- Is explicitly defined and used in the kernel code
- One in each SM
- Accessed at much higher speed (in both latency and throughput) than global memory

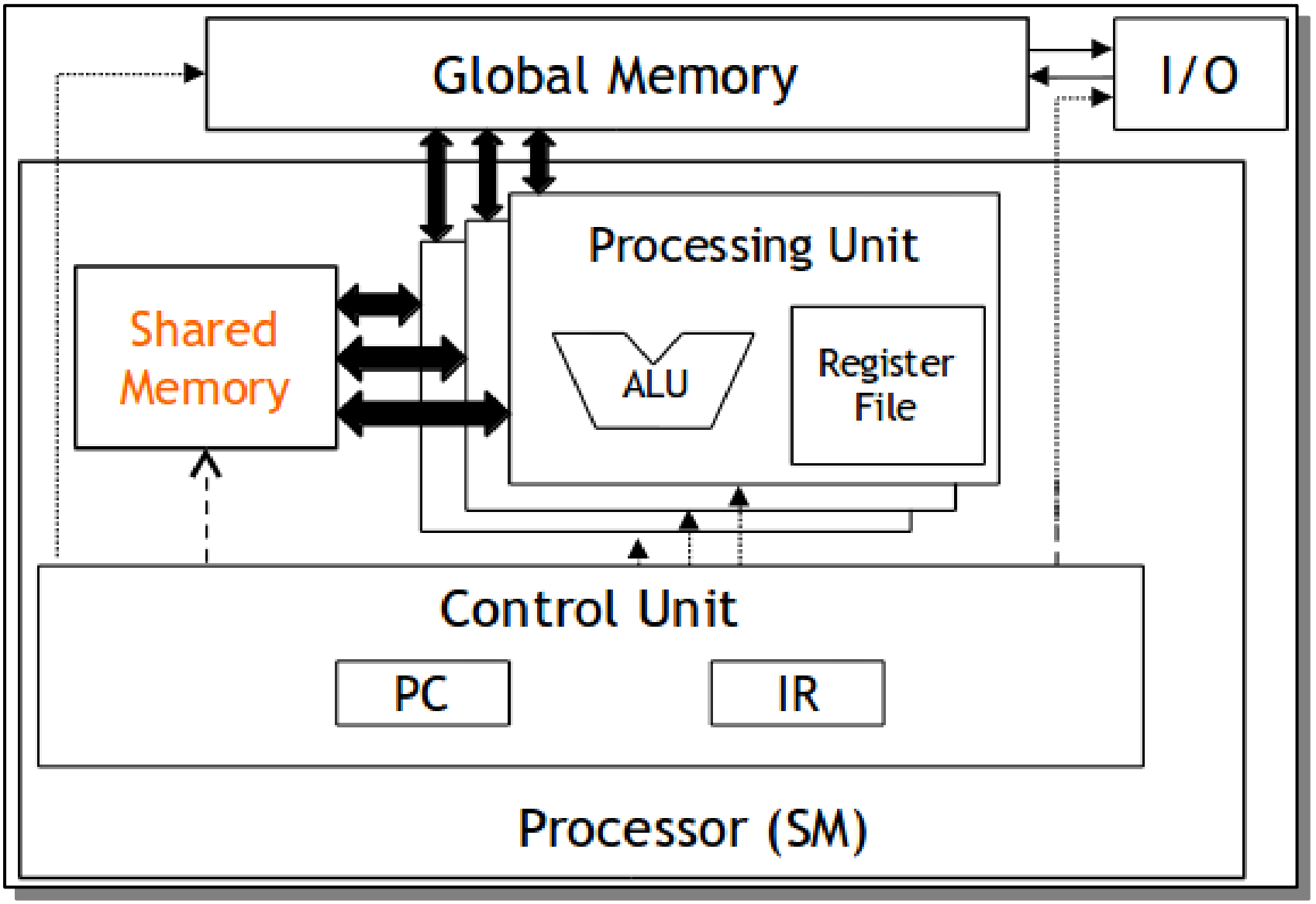
Shared Memory in CUDA

- Is explicitly defined and used in the kernel code
- One in each SM
- Accessed at much higher speed (in both latency and throughput) than global memory
- Scope of access and sharing - thread blocks

Shared Memory in CUDA

- Is explicitly defined and used in the kernel code
- One in each SM
- Accessed at much higher speed (in both latency and throughput) than global memory
- Scope of access and sharing - thread blocks
- Accessed by memory load/store instructions

Hardware View of CUDA Memories



Programming Strategy

- Partition data into subsets or tiles that fit into shared memory

Programming Strategy

- Partition data into subsets or tiles that fit into shared memory
- Use one thread block to handle each tile

Programming Strategy

- Partition data into subsets or tiles that fit into shared memory
- Use one thread block to handle each tile
  - Load the tile from global memory to shared memory, in parallel (using multiple threads)

Programming Strategy

- Partition data into subsets or tiles that fit into shared memory
- Use one thread block to handle each tile
  - Load the tile from global memory to shared memory, in parallel (using multiple threads)
  - Perform computation on the subset in shared memory

Programming Strategy

- Partition data into subsets or tiles that fit into shared memory
- Use one thread block to handle each tile
  - Load the tile from global memory to shared memory, in parallel (using multiple threads)
  - Perform computation on the subset in shared memory
    - (Reduces traffic to the global memory)

Programming Strategy

- Partition data into subsets or tiles that fit into shared memory
- Use one thread block to handle each tile
  - Load the tile from global memory to shared memory, in parallel (using multiple threads)
  - Perform computation on the subset in shared memory
  - Upon completion, write results from shared memory to global memory

Shared Memory Variable Declaration

```
__global__
void MatrixMulKernel(int m, int n, int k, float* A,
float* B, float* C)
{
   __shared__ float ds_A[TILE_WIDTH][TILE_WIDTH];
   __shared__ float ds_B[TILE_WIDTH][TILE_WIDTH];

}
```

Device Query

```
int dev_count;
cudaGetDeviceCount(&dev_count);
```

Device Query

```
int dev_count;
cudaGetDeviceCount(&dev_count);
```

```
cudaDeviceProp dev_prop;
for (i = 0; i < dev_count; i++) {
    cudaGetDeviceProperties(&dev_prop, i);
    // decide if device has sufficient
resources and capabilities
}
```

Device Query

```
int dev_count;
cudaGetDeviceCount(&dev_count);
```

```
cudaDeviceProp dev_prop;
for (i = 0; i < dev_count; i++) {
    cudaGetDeviceProperties(&dev_prop, i);
    // decide if device has sufficient
resources and capabilities
}
```

```
cudaDeviceProp is a built-in C structure type
dev_prop.dev_prop.maxThreadsPerBlock
Dev_prop.sharedMemoryPerBlock
```