Virtual Workshop

Introduction to GPGPU and CUDA Programming: Memory Architecture

A CUDA device has a number of different memory components that are available to programmers - register, shared memory, local memory, global memory and constant memory. Figure 6 illustrates how threads in the CUDA device can access the different memory components. In CUDA only threads and the host can access memory.

Memory Access

We use one and two-way arrows to indicate read (R) and write (W) capability. An arrow pointing toward a memory component indicates write capability; an arrow pointing away from a memory component indicates read capability. For example, global memory and constant memory can be read (R) or write (W).

On a CUDA device, multiple kernels can be invoked. Each kernel is an independent grid consisting of one or more blocks. Each block has its own per-block shared memory, which is shared among the threads within that block. All threads can access (R/W) different parts of memory on the device that are summarized below:

In general, we use the host to:

- Transfer data to and from global memory
- Transfer data to and from constant memory

Once the data is in the device memory, our threads can read and write (R/W) different parts of memory:

- R/W per-thread register
- R/W per-thread local memory
- R/W per-block shared memory
- R/W per-grid global memory
- R per-grid constant memory

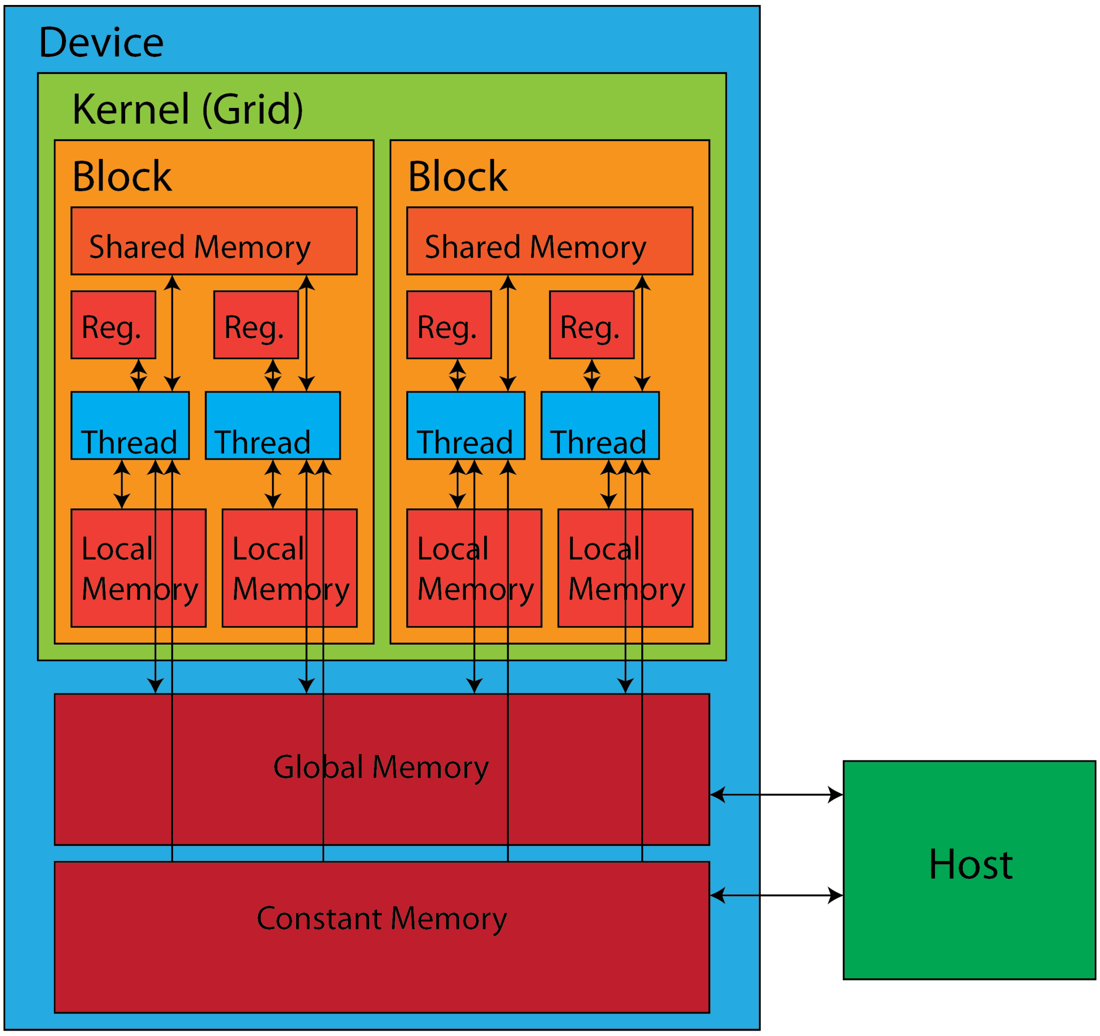

Figure 6

Size and Bandwidth

Different memory components have different size and bandwidth. Global memory has the lowest bandwidth but largest memory size (5Gb on the K20 GPUs found on Stampede). The constant memory allows read-only memory access by the device and provides faster and more parallel data access.

Per-block shared memory is faster than global memory and constant memory, but is slower than the per-thread registers. Each block has a maximum of 48k of shared memory for K20. Per-thread registers can only hold a small amount of data, but are the fastest. Per-thread local memory is slower than the registers and is used to hold the data that cannot fit into the register.

Qualifiers

CUDA supports different types of qualifiers to assign variables to different memory components. Different memory components also have a life span. The following table (see the CUDA Programming Guide) summarizes the scope, lifespan, and memory type for different qualifiers.

Qualifiers	Location	Scope	Lifespan
Automatic variable	Register	Thread	Kernel
Automatic array	Local	Thread	Kernel
__device__ , __shared__ , int SharedVar	Shared	Block	Kernel
__device__ , int GlobalVar	Global	Grid	Application
__device__ , __constant__ , int ConstVar	Constant	Grid	Application

Automatic Variables

An automatic variable is declared without any qualifiers. It resides in the per-thread register and is only accessible by that thread.

```
int autovar;
```

Automatic Arrays

An automatic array variable resides in the per-thread local memory and is only accessible by that thread. However, it is possible for the compiler to store automatic arrays in the registers, if all access is done with constant index values. An automatic array can be declared in the following manner:

```
int autoarr[];
```

__shared__

Appending the __*shared*__ variable type qualifier explicitly declares that the variable is shared within a thread block. You can append an optional __*device*__ variable type qualifier to achieve the same effect.

```
__shared__ int shvar;

//is the same as

__device__ __shared__ int shvar2;
```

__device__

When the __*device*__ qualifier is used by itself, it declares the variable resides in global memory.

```
__device__ int dvvar;
```

__constant__

Appending the *__constant__* qualifier declares a constant variable that resides in the constant memory. You can also append an optional *__device__* to achieve the same effect. A constant variable must be declared outside of any function body.

```
__constant__ int cnvar;

//is the same as

__device__ __constant__ int cnvar2;
```

Assign by the address

You can also assign the address of a variable declared in the shared, constant, or global memory to a pointer variable.

```
float *ptr= &GlobalVar;
```

See the appendix for more information about the variable type qualifiers.

<= previous next =>

Add my notes
Mark (M) my place in this topic