Memory affects the execution of threads
Memory affects the execution of threads

Thread blocks are scheduled by the hardware on multiprocessors and more than one block can be simultaneously scheduled on a multiprocessor, thus sharing its resources, particularly shared memory and registers
Memory affects the execution of threads

Thread blocks are scheduled by the hardware on multiprocessors and more than one block can be simultaneously scheduled on a multiprocessor, thus sharing its resources, particularly shared memory and registers

So the pattern of use of shared memory can put a limit on the number of blocks in the grid
Memory affects the execution of threads

Thread blocks are scheduled by the hardware on multiprocessors and more than one block can be simultaneously scheduled on a multiprocessor, thus sharing its resources, particularly shared memory and registers

So the pattern of use of shared memory can put a limit on the number of blocks in the grid

Similarly, there is a limit on the number of threads per block: up to 65536 in the above
GPUs offers a huge amount of processing power at low cost, but in a way that is extremely sensitive to memory access.
GPUs offers a huge amount of processing power at low cost, but in a way that is extremely sensitive to memory access.

It is easy to get started with CUDA as it is basically C, but you do have to be very aware of the properties of memory.
Modern GPUs support *unified memory spaces*
Modern GPUs support *unified memory spaces*

This allows you to use a single address space for both host and device memory and not worry which is which.
Modern GPUs support *unified memory spaces*

This allows you to use a single address space for both host and device memory and not worry which is which.

A hidden mechanism copies data between CPU and GPU as necessary.
Modern GPUs support *unified memory spaces*

This allows you to use a single address space for both host and device memory and not worry which is which

A hidden mechanism copies data between CPU and GPU as necessary

Exercise. Is this a good idea?
Current hardware has limits

- Floating point is mostly single precision and deviates slightly from IEEE; cards supporting double precision are much more expensive
- Block shared memory is too small
- Warp size perhaps could be smaller. Or larger. Or variable.
- Access to global memory is extremely slow
Current hardware has limits

- Floating point is mostly single precision and deviates slightly from IEEE; cards supporting double precision well are much more expensive
Current hardware has limits

- Floating point is mostly single precision and deviates slightly from IEEE; cards supporting double precision well are much more expensive
- Block shared memory is too small
Current hardware has limits

- Floating point is mostly single precision and deviates slightly from IEEE; cards supporting double precision well are much more expensive
- Block shared memory is too small
- Warp size perhaps could be smaller. Or larger. Or variable.
Current hardware has limits

- Floating point is mostly single precision and deviates slightly from IEEE; cards supporting double precision well are much more expensive
- Block shared memory is too small
- Warp size perhaps could be smaller. Or larger. Or variable.
- Access to global memory is extremely slow
These issues mean that to get the best performance programs have to be tuned for the specific GPU card they are running on.
These issues mean that to get the best performance programs have to be tuned for the specific GPU card they are running on. This makes portability more of an issue than we’d like.
These issues mean that to get the best performance programs have to be tuned for the specific GPU card they are running on. This makes portability more of an issue than we’d like. Even between GPUs from the same manufacturer.
A couple of notes on floating point support in GPUs:

Consumer cards support single precision (32 bit) floats well and double precision (64 bit, the normal size for CPUs) poorly. For example, some recent Nvidia cards (e.g., GTX 1080) the double precision throughput is 1/32 that of single precision. This is because video games don't need the precision and are more interested in speed ("get it wrong quickly"). For scientific computing, double precision is necessary, so only expensive cards support that.
A couple of notes on floating point support in GPUs:

Consumer cards support single precision (32 bit) floats well and double precision (64 bit, the normal size for CPUs) poorly.
A couple of notes on floating point support in GPUs:

Consumer cards support single precision (32 bit) floats well and double precision (64 bit, the normal size for CPUs) poorly.

For example, some recent Nvidia cards (e.g., GTX 1080) the double precision throughput is 1/32 that of single precision.
A couple of notes on floating point support in GPUs:

Consumer cards support single precision (32 bit) floats well and double precision (64 bit, the normal size for CPUs) poorly.

For example, some recent Nvidia cards (e.g., GTX 1080) the double precision throughput is 1/32 that of single precision.

This is because video games don’t need the precision and are more interested in speed (“get it wrong quickly”).
A couple of notes on floating point support in GPUs:

Consumer cards support single precision (32 bit) floats well and double precision (64 bit, the normal size for CPUs) poorly.

For example, some recent Nvidia cards (e.g., GTX 1080) the double precision throughput is 1/32 that of single precision.

This is because video games don’t need the precision and are more interested in speed (“get it wrong quickly”).

For scientific computing, double precision is necessary, so only expensive cards support that.
The latest trendy application, AI using *deep learning*, needs even less precision: *half precision* (16 bit)
The latest trendy application, AI using *deep learning*, needs even less precision: *half precision* (16 bit)

So GPUs are starting to support that, too
The latest trendy application, AI using *deep learning*, needs even less precision: *half precision* (16 bit)

So GPUs are starting to support that, too

Presumably with the same kind of market segmentation on consumer and professional products
Next, there is the extra problem shared by all coprocessors: memory bandwidth between the main CPU and the coprocessor.
Next, there is the extra problem shared by all coprocessors: memory bandwidth between the main CPU and the coprocessor.

Copying data in and out of the GPU is significantly time consuming.
Topics

GPUs

Next, there is the extra problem shared by all coprocessors: memory bandwidth between the main CPU and the coprocessor.

Copying data in and out of the GPU is significantly time consuming.

So we need to worry about data movement between the GPU and the main CPU.
Next, there is the extra problem shared by all coprocessors: memory bandwidth between the main CPU and the coprocessor.

Copying data in and out of the GPU is significantly time consuming.

So we need to worry about data movement between the GPU and the main CPU.

And, if possible, overlap data transfers with GPU and CPU computation.
Next, there is the extra problem shared by all coprocessors: memory bandwidth between the main CPU and the coprocessor.

Copying data in and out of the GPU is significantly time consuming.

So we need to worry about data movement between the GPU and the main CPU.

And, if possible, overlap data transfers with GPU and CPU computation.

And overlap CPU and GPU computations.