Introduction to Parallel Programming (CS344)

By Erik Pärlstrand

Lesson 1: The GPU Programming Model

Assume that we want to build faster processor. The following options is available:

1. Run with a faster clock speed (i.e. shorter amount of time on each step of computation).
2. Do more work per clock cycle.
3. Add more processes.

In this course we will take approach number three and in particular we are interested in GPU (Graphical Processing Unit). Recent trends is that the clock rates for transistors does not increase and therefore need to increase the number of cores.

Latency versus throughput (i.e. bandwidth)

- CPUs are low latency, low throughput.
- GPUs are higher latency, high throughput.

By latency we mean "time to respond" while throughput is computation/time. Analogy is that the CPU is a Ferrari but the GPU is a bus.

GPU design tenets

1. Many simple compute units, little control logic.
2. Explicitly parallel programming model (unlike many CPU compiler toolchains that makes the hardware details a bit more opaque, especially across different CPU hardware).
3. Optimize for throughput instead of latency.

CUDA

In Cuda one usually differs between:

- Host: part of program runs on the CPU.
- Device: part of program that runs on the GPU.
- Assume host and device have separate memories to store data.

Note that the CPU is "in charge", i.e. it's responsible for:

- Move data from CPU memory to GPU.
- Move data from GPU back to CPU.
- Allocate GPU memory.

- Launch kernel on GPU .

In cuda we write kernels (a function to run on GPU) as a serial program. Parallelism is introduced by the CPU telling the GPU to launch multiple instances (i.e. threads) of the kernel. GPU's are good at launching a lot of threads and running them in parallel.

Threads and blocks

The syntax for specifying blocks of threads in cuda is e<<<B, T>>>, where B is the number of blocks and T is the number of threads. There is a maximum number of threads per block (512 for older GPUs, 1024 for newer).

For example, in order to launch 128 threads we can either launch 2 blocks with 64 threads each or 1 block with 128 threads. Note that each thread knows the index of its block and thread.

Note that in the previous example we used 1 dimensional block/threads, however we can define them in three dimensions, i.e. B = dim3 (x,y,z) and T = dim3 (x,y,z). Below is an example of the organization of threads within a block and how they create grids: