TDT4200 Parallel Computing
Problem Set 5

Stian Hvatum (hvatum)
MTDT

October 22, 2012

Contents

Task 1 **Theory** **1**
1. Differences between CPUs and GPUs 1
2. Problems better suited to GPUs 1
3. Problems better suited to CPUs 2
4. Why CPU might be faster 2

Task 1 Theory

1. Differences between CPUs and GPUs

While a CPU mostly is a general purpose computational unit, the GPU is specialized in rendering graphics. Since graphics often benefit from a highly parallel architecture, GPUs usually have hundreds of simple and slow cores. The simple and slow cores use little space, and thus there can be many of them. The GPU usually has a faster memory than a normal CPU, since its may cores needs rapid memory access, eg. for textures and vertex location.

Another major difference is that a CPU often connect to a lot of peripherals, has a lot of different interfaces, and needs to speak a lot of protocols. A GPU is mostly connected to the CPU via a PCI Express bus[1] or similar, it's own memory and maybe some other GPUs[2].

2. Problems better suited to GPUs

The key for problems that will be more efficiently handled by a GPU is that they can be highly parallelized. Rendering graphics is a typical application, since it consists

[1]Which is actually no bus at all, only named such by marketers. PCI Express is a point-to-point connection
[2]like SLi or CrossFireX

of a lot of small but equal task. First you have a lot of vertices, which for each one we need to transform into primitives. Then, for each primitive we must generate geometry, and last we rasterize the primitives on a single buffer. These tasks can all be parallelized, and thus the reason for why GPUs work the way they do, and a normal CPU would need to do a lot of this sequentially[3]. Another possible, more general use, is to solve mathematical equations that use a lot of vectors and matrices. With some clever math and indexing, you can effectively parallelize a lot of eg. Linear Algebra Systems, matrix multiplication and so on. This is faster by the same argument as graphics, all the simple computations that can be effectively parallelized.

3. Problems better suited to CPUs

Problems with a lot of branches, or that simply is not possible to parallelize effectively, is usually better to just run on the CPU. A GPU will, because of it's SIMD nature, execute all paths of a branch, only the "wrong" path will be filled with NOPs. This is because it cannot know wherever the condition is true or false for all threads, and they share a common program counter. A GPU will typically also have trouble with recursion.

A problem that would be more effectively solved on a CPU rather than a GPU may be compiling. Compiling code means a lot of both branching, and possibly also recursion. It is also a big problem for the GPU and parallelization in general if there is a strong data dependency. A typical example where data dependency is obvious, is the calculation of Fibonacci numbers, where each number depends on the two later ones.

4. Why CPU might be faster

While some problems are way faster to execute on a GPU, the GPU is still an external device that needs special calls and handling both for feeding data and for returning data. Usually, GPUs are connected to the CPU via PCI Express. The PCI Express is a fast connection in terms of peripheral connection, but is still very slow compared to the internal paths of the CPU. Because of this, copying the problem data to the GPU, compute it really fast, then transferring the results may be slower than computing the problem slowly on the CPU. This, of course, depends on many things, like the amount of data that needs to be transferred, the difference in execution speed on GPU/CPU, and the connection speed between CPU/GPU and CPU/Main Memory.

[3]as the number of vertices/primitives/geoms way outnumbers the number of cores in a general CPU