Mid-Term Report

Our research into GPU compilers has surfaced quite a few papers on what sort of optimizations are done inside a compiler for a GPU. This includes topics such as:

- Kernel optimizations
- Memory optimizations
- Vectorization
- Communication
- Loop unrolling

Along with these, our report will be answering questions of why it’s important to optimize these features and what sort of speedup GPU programs gain.

Upon further research into the topic of GPU compilers, we have found that there have been more emphasis on rather implementing new languages that take advantage of the GPU, implement source-to-source compilers which take programs such as C/C++ and translate them into CUDA/OpenCL programs. The reasoning behind this movement is that programmers do not take advantage of the GPU due to the fact that they have to learn a new language in order to do so, rather than just a new technique within a language that they already know.

If time allows, we may even dive into CPU-GPU hybrid compilers. At the moment, it seems as if we have enough subject matter to play around with and won’t be able to get around to it.

Potential Paper Outline

**Introduction**
- Why are optimizations important?
- Why is GPU programming important?
- Why would be taking advantage of a GPU be advantageous?

**Brief Background**
- General layout of the GPU (perhaps nvidia’s maxwell? or some recent architecture)
  * perhaps compare to modern CPU?
- What threads are and how they work in a GPU
- How memory works in a GPU
- What kernels (essentially functions) are and how are they used

**Memory optimizations**
- Go into detail why memory is a big issue with GPUs
- How is it done
  * Global memory
  * Local memory
  * Private memory
  * Image memory
  * Constant memory
- Vectorization
- Code examples?

**Loop unrolling**
- Types of unrolling
- Why it's done
- Code examples?

**Source-to-source Compilers**
- Why is it done?
- Advantages/disadvantages

**Hybrid CPU-GPU Compilers (If we have time)**

**Applications**
- Perhaps show some examples of what GPU programming can accomplish

**Sources**

Optimizing general purpose compiler optimization

Compiler Transformations for High-Performance Computing

The Implementation of a High Performance GPGPU Compiler

An OpenMP Compiler for Hybrid CPU/GPU Computing Architecture

A compiler for high performance computing with many-core accelerators


Rak, A.; Feldhoffer, G.; Soos, B.G.; Cserely, G., "CPU-GPU hybrid compiling for general purpose: Case studies," Cellular Nanoscale Networks and Their Applications (CNNA), 2010 12th International Workshop on, vol., no., pp.1,1, 3-5 Feb. 2010
