

CS675 GPGPUs versus Multi-core CPUs

Wim Bohm, Computer Science Department, CSU

In his famous 1993 paper "*Twelve ways to fool the masses when giving performance results on Parallel Computers*", and again in a 2009 update "*Misleading Performance Claims in Parallel Computations*", David H. Bailey warns against non-scientific, hyped claims about the performance of certain novel parallel computing systems as compared to more conventional computing systems. He states that "Scientific research must be based on solid empirical data and careful, objective analysis of that data", and that "Scientists must be willing to provide all details of the experimental environment, so others can reproduce their results." One common way to *fool the masses* is to compare highly optimized code on one system against un-optimized code on another.

The recent advent of GPGPUs (General Purpose computing on GPUs) appears to be a case in point, where we need to be particularly vigilant. It is true that GPUs provide significantly more compute resources than comparable CPUs. This power comes in the form of "*massive multi-threading*." However, we are beginning to see what we believe are exaggerated claims about the benefits of GPGPU.

For example, Rizk and Rajopadhye at CS CSU were able to speed up the UNAFold RNA secondary structure package (for RNA sequences with 10K nucleotides) on an NVidia Tesla 1060, by a factor of 130 over the publically released (but naive and sequential) C code. They achieved this through a number of sophisticated optimizations including algorithmic reordering, tiling, and exploiting medium grain parallelism. This seems impressive until you take stock of the fact that many of these techniques can also be used to optimize the sequential code. Doing so yields an improvement by an order of magnitude for the CPU. Moreover, exploiting the multi-core architecture of current CPUs and multi-thread our CPU code could win back another factor of 4 or 8 for the CPU side.

So how do the CPU and the GPU finally compare? We don't know yet and hope to discover this by careful study in the class.

In this Spring 2010 version of CS675 we want to do thorough, unbiased comparison studies of GPUs versus multi-core CPUs. In particular, we would like to study differences in the Model of Computation for these systems. Both are based on multi-threading, but GPU thread blocks are completely independent of each other, whereas CPU threads can share data and synchronize. We want to find out how this influences the style of programming.

We plan to study the expressive power of CUDA (and OpenCL), and its machine dependence. We will study how the number of multi processors or the number of memory banks in global or shared memory, is reflected in highly efficient code, thereby rendering it machine dependent. We will study the steps towards highly efficient codes in both GPUs and multi-cores and establish the trade off between performance and programming effort. Certain techniques will be useful in both models: vectorization (coalescing in GPUs), tiling for locality, data dependence analysis and scheduling.

This hands-on course will start with studying "micro benchmarks": very small codes establishing one aspect of the performance of a system, eg how to use tiling and SSE instructions to speed up matrix multiply, or how to get the highest possible GFLOPS/sec rate or memory bandwidth out of a GPU, and how this compares with quoted maximum machine rates (theoretical machine peak).

We will then move to the implementation of dense linear algebra, vision (stencils, convolutions, pyramids) and bio-informatics applications (dynamic programming).