Efficiently Generating Code for the GPU

Matthew Le

Clock speeds in microprocessors are no longer increasing at the rate that they did years ago due to power consumption constraints and heat dissipation. In order to increase the speed of computation, multicore computing must be exploited. Graphics Processing Units (GPU's) are used for image rendering and optimized for heavy parallelized use, and are therefore ideal candidates. Many GPU's come with as many as 3,000 cores [1], which in addition to image rendering, can be used for general purpose computation. Despite the existence of programming languages such as CUDA and openCL, that allow a programmer to run general purpose code on the GPU. Current solutions suffer from two major limitations: First, it is a complex and error-prone process because the programmer is expected to manually manage memory as well as the parallel threads. Second, transferring data to the GPU is costly, therefore, it is not always optimal to offload code to the GPU as the data transfer cost might outweigh the parallel gains.

Efficiently and creatively increasing computation speed is of paramount importance, especially given the rapid increase amounts of data. Increasing data directly implies more computation, and being able to effectively parallelize code involving large data sets is necessary for many data analysis applications in the climate and life sciences (amongst others). An interesting research avenue that Big Data brings to the table is that there are different ways to parallelize data, because it can be broken up in so many different ways.

Traditionally, a compiler reads in the source code of a program, performs semantic analysis and then generates machine-readable code. While performing this semantic analysis, the compiler collects information about the program to identify opportunities to generate parallel code. However, at compile-time, not all information is available to produce the most efficient code, as useful information, such as the data's size and dimensionality, is only available at run-time.

This research aims to exploit both compile-time (static) and run-time (dynamic) analysis to generate code that can be offloaded to the GPU efficiently. In cases where GPU offloading is not beneficial, we would also like to determine what is the best alternative (*e.g.* parallelize on CPU, loop unrolling, *etc.*) In cases where it is beneficial, we will explore what is the best way to split up the data for parallelization on the GPU. Since factors that determine the utility received from running code on the GPU can not all be determined statically, I am proposing that static analysis in conjunction with dynamic analysis can aid in the effort of generating efficient code for the GPU. The purpose of this research is not to focus on finding more opportunities to parallelize code, rather it is to find more efficient ways of parallelizing code that is already known to be able to be run in parallel.

There are two main components to this approach: the compiler and the runtime system. The compiler first determines the parallel regions of code, and given the limited information it has access to, it will generate multiple code scenarios to account for such uncertainty including the possibility of not offloading the code to the GPU at all. The compiler then passes on these scenarios to the runtime system, which then chooses the best scenario based on the data's size and dimensionality that becomes available only at runtime. Overall, the scenario generation process adds a negligible amount of computation at compile-time, while producing significant savings at runtime.

The main goal of this research is to create a compiler optimization that will be able to parallelize code in the most efficient way possible. To be clear, we are not searching for more opportunities to parallelize code, but exploring different ways to parallelize code that we know can already be run in parallel. Relieving the burden of writing parallel code from the programmer makes writing computationally intensive applications much easier. For instance, a large number of climate scientists use computer models to project future climate. These simulations take anywhere from weeks to months to run and, therefore, make exploratory research very costly. My research would allow programmers from other domains who may not have a wealth of experience writing code to be able to create applications that can run quickly and efficiently, effectively allowing them to spend their time uncovering facts of tremendous scientific and societal importance as opposed to optimizing computer code.

There are two main projects that this research draws from. The first project is known as DyManD (Dynamically Managed Data)[2], where they allow the programmer to write code that runs on the GPU, and their compiler takes care of memory management between the CPU and GPU. What sets this research apart is that we are aiming to abstract all implementation of parallel algorithms from the programmer, which their research effort does not address. Also, they are not altering how the data is parallelized. All of this is up to the programmer to decide. Halide[4] is another project that has addressed the issue of simplifying parallel programming. Halide is a domain specific language for image processing, which allows the programmer to write parallel code at a high level of abstraction, and specify how the code is parallelized. However, with Halide, the programmer must still know about the implementation of parallel algorithms. Also, this language is restricted to the image processing domain, so it has limited advantages to general purpose parallel programming.

It is my hope that I am given the opportunity to join the Liberty research group at Princeton University, where the work of Prof. David August in highly relevant to my proposal. I believe that Prof. August's mentorship and expertise in parallelizing techniques would allow me to successfully carry out my research project in a timely manner. Additionally, Prof. August's tremendous teaching record will be a great example for me on how to carry out my research expertise to the classroom and to the general public.

References

[1] GeForce GTX 690. "World's Fastest Graphics Cards, GPU's, and Video Cards. N.p, n.d. Web. 10 Oct. 2012. <http://www.geforce.com/hardware /desktop-gpus/geforce-gtx-690/specifications>

[2] T.B. Jablin, J.A. Jablin, P. Prabhu, F. Liu, D.I. August. Dynamically Managed Data for CPU-GPU Architectures. In *Proceedings of the 2012 Internations Symposium on Code Generation and Optimization (CGO)*, March 2012.

[3] J.T. Overpeck, G.A. Meehl, S. Bony, D.R. Easterling. Climate Data Challenges in the 21st Century. In *Science Magazine*, February 11, 2011. Volume 331

[4] J.R. Kelley, A. Adams, S. Paris, M. Levoy, S. Amarashinghe, F. Durande. Decoupling Algorithms from Schedules for Easy Optimization of Image Processing Pipelines. In *ACM Transaction on Graphics - SIGGRAPH 2012 Conference Proceedings*, Volume 31 Issue 4, July 2012