2011 IEEE International Parallel & Distributed Processing Symposium

A Framework for Automated Performance Tuning and Code Verification on GPU Computing Platforms

Allison S. Gehrke, Ilkyeun Ra, Daniel A. Connors
Dept. of Computer Science and Engineering and Dept. of Electrical Engineering
University of Colorado Denver
Denver, CO 80204 U.S.A
{*Allison.Gehrke, Ilkyeun.Ra, Dan.Connors*}*@ucdenver.edu*

***Abstract*—Emerging multi-core processor designs create a computing paradigm capable of advancing numerous scientific areas, including medicine, data mining, biology, physics, and earth sciences. However, the trends in multi-core hardware technology have advanced far ahead of the advances in software technology and programmer productivity. For the most part, current scientists only leverage multi-core and GPU (Graphical Processing Unit) computing platforms after painstakingly uncovering the inherent task and data-level parallelism in their application. In many cases, the development does not realize the full potential of the parallel hardware.**

There exists an opportunity to meet the challenges in optimally mapping scientific application domains to multi-core computer systems through the use of compile-time and link-time optimization strategies. We are exploring a code compilation framework that automatically generates and tunes numerical solver codes for optimal performance on graphical processing units. The framework advances computational simulation in kinetic modeling by significantly reducing the execution time of scientific simulations and enabling scientists to compare results to previous models and to extend, modify, and test new models without code changes.

***Keywords*-high performance computing (HPC); auto-tuning; code generation; scientific computing; graphical processing unit; numerical simulation**

I. Introduction

The focus of this dissertation research is on building a code compilation framework that automatically generates and tunes numerical solver codes for optimal performance on graphical processing units. Preliminary research indicates that current GPU compilers can perform more aggressive use of on-chip registers and shared memory for data-level parallel applications. Initial experiments show an increase in GPU performance by up to 20% without involvement of the application programmer. Simply stated, a scientist should be concerned with the details of their respective model and is not trained to exploit the full architectural resources and capabilities of an advancing computing machine. Moreover, often the exploitable resources, such as processor registers and on-chip programmer-controller memories, scale with each new generation of multi-core system.

The computationally intensive numerical simulation of ion channel kinetics and optimization of kinetic model rate constants (fits to experimental data) is used as a target for evaluating our code generation system. The kinetics application uses a genetic algorithm to stochastically search parameter space to find global optima. As each individual in the population describes a rate constant parameter set in the kinetic model and the model is evaluated for each individual, there is significant computational complexity and parallelism in even a simple model run.

II. Problem Description and Proposed Approach

High performance computing (HPC) modeling, simulation, and optimization capabilities are key to accelerating scientific discovery in many disciplines. Numerical simulations are also indispensable in industrial production and manufacturing. Simulations of complex models that span wide temporal and spatial scales requires computationally demanding numerical algorithms and optimization methods that have execution times that can range from hours to weeks, even on high performance computers. Naturally, science and industry are very interested in how to significantly reduce simulation run times. Two of the biggest challenges in HPC research are how to efficiently build applications that achieve a reasonable fraction of the theoretical peak performance available on HPC systems and how to democratize research capabilities to bring the benefits of HPC to the masses.

Our goal is to construct a code generation framework that allows scientists to describe their model details in an abstract intermediate representation. The framework will then analyze the model description and make register and shared memory storage assignments. The framework will thereby overcome the generational performance drift associated with evolving GPU platforms. In addition, we are exploring capabilities within the code generation framework to automatically generate performance timing and scientific model data collectors. These capabilities will aid computational scientists in using parallel programming models to more transparently leverage HPC capabilities and thereby develop new scientific models quickly and accurately.

III. Significance of the Research

We developed an efficient GPU architecture implementation of the implicit Runge Kutta 4th order method (RK4) for solving systems of ordinary differential equations (ODEs) that is generally applicable to other scientific applications. These advancements enable researchers to interact more easily with their simulations. Neuroscientists can now validate existing and alternative kinetic models, conduct sensitivity analysis studies, test a broader range of parameter sets to establish a reasonable range of parameter values, and to conduct statistical comparison studies. As a result, we demonstrate ion channel models that more completely describe all key biological characteristics of AMPA receptors.

A. *Auto-tuning for Performance Framework*

This thesis demonstrates that an automated compile-time code generation system can construct tailored mappings of scientific numerical algorithms to modern multicore architecture resources. Automation of code increases scientific productivity by discovering optimal performance mapping for different scientific models, eliminating the necessity for scientists to labor intensively on inessential architectural details and by assisting verification and validation efforts of new scientific models.

Fig. 1 illustrates the main components of the framework and how they interact. The framework supports semi-automatic code generation of GPU architectures by exploiting on-chip register and shared memory resources. The framework enables the investigation of how the performance of the program scales as the available resources and parallelism change. In addition, it automates the time-intensive and error prone process of performance optimization.

A primary motivation for auto-tuning is to keep up with rapidly innovating hardware without having to change your code base [1]. There are generally two methodologies for auto-tuning, model-driven optimization which has it's origins in compiler research and empirical optimization. Empirical optimization techniques generate parameterized code variants for a given algorithm [1] and searches over the optimizations to minimize runtime. We employ an empirical auto-tuning optimization methodology.

The scientific model is input as an abstract intermediate representation. The framework analyzes the model description and makes register and shared memory storage allocations based on the per thread storage requirements of the model. The collection of timing data or model data is configurable (can be turned on or off) as well as running an optimized simulation vs running verification routines or tuning for a new model.

The framework generates many different code versions for each GPU configuration. Timing data and compiler resource allocation information from the PTX (Parallel Thread eXecution) files is collected. PTX is similar to assembly code on CPUs. We will use a PTX analyzer to determine the register and shared memory allocations the compiler made in order to make more aggressive and application-educated optimization choices. We also use a GPU simulator to explore memory accesses and to artificially derive other points based on auto-tuning to examine hardware limits.

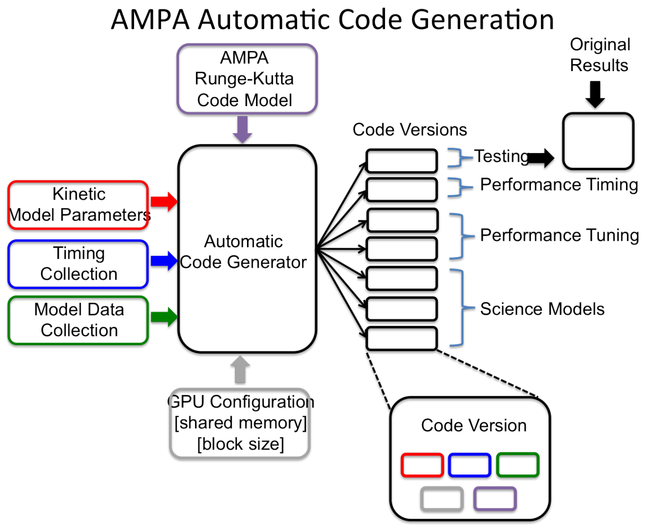

Figure 1. Automatic Code Generation Framework. This figure illustrates the different components of the framework and how they interact. Kinetic models are evaluated for memory requirements. Code for different GPU configurations are auto-generated for test and evaluation. Users can collect performance tuning data, timing data, and/or model data.

There are many optimization strategies (parameters to tune) for each target architecture. We are focusing on the following for the GPU: shared memory to reduce global memory access; unroll loops to avoid arrays in the kernel to reduce thread divergence and to take advantage of register resources to reduce access to local memory; and experimenting with different block sizes to determine optimal configurations.

B. *Auto-Verfication*

To improve scientific productivity, we are exploring the ability to detect coding errors that can be stubbornly insensitive to the solution (verification) and that transparently extracts, examines, and compares scientific data between existing and emerging implementations (validation).

Verification of ODE solvers naturally includes a determination that the solver has solved the equations correctly [2]. One way to do this is a systematic convergence study (described in introductory numerical methods textbooks) that verifies if the solution converges at the expected rate for a given numerical method and that the error approaches zero [2]. However, Kinetic modelers and modelers in general, are not necessarily versed on the inherent traps in numerical simulation and may never see the need to use time and resources to test their code if the framework to do so automatically is not available.

Other methods we are exploring is inter-code comparisons where a given code is verified against known analytical solutions and the answers from each are checked for agreement to within the iterative convergence tolerance ([3], [4]) and by comparing the code results with data from past experiments as done in a large computational simulation project profiled in [5]. Ironically, the historical reason convergence studies and inter-code comparisons are not routinely integrated in simulation studies are because of limited computer resources and the effort it takes to develop multiple solutions.

C. *Summary*

The overall results demonstrate that the automated process can improve performance of a target application and that performance improvements are sustained across scientific models and across GPU hardware generations. The increase in performance opens the door for new applications. We show that an automated fitting approach can be applied to statistical comparison studies, an idea considered impractical by [6]. The work enables rapid design of complicated kinetic models that more accurately represent what occurs in nature that could not be considered before due to computational limitations. Examining applicability to other scientific applications that use the RK4 method further proves these contributions.

In summary, this thesis determines which parallel strategy and architecture is best to achieve scalable performance for advanced numerical simulations and integrates verification and validation into the automation framework. Altogether the framework promotes a fundamental change in computational science research and promotes HPC adoption to a broader range of researchers that could potentially use highly parallel architectures.

IV. Related Work

The literature tends to narrowly focus on specific building blocks like basic linear algebra algorithms and numerical solvers but seldom address the complex development process for high performance computing applications comprehensively [7]. Optimizing kinetic models on emerging computer architectures is unusually difficult because expertise in chemical kinetics and modeling must be combined with a strong understanding of numerical simulation and various parallel and computer architecture paradigms [7].

There needs to be more focus on the accessibility of the technology. Most modeling tools that kinetic modelers use enable the investigation of a rate constant set for a single model ([8], [9]). A. Robert and J. R. Howe [6] used Monte Carlo simulations with a software program called ChannelLab (synaptosoft, Inc.) and then manually compared thousands of different solutions to experimental data to find a good fit. Optimization routines (in the mathematical sense, the maximization or minimization of an objective function) and optimized performance support on multicore and high performance computing systems are not included.

Ultimately, this dissertation hopes to make a substantial contribution to the efforts of the Kinetic Preprocessor (KPP) [10] and NWChem software. KPP was designed as a general analysis tool to aid simulation of chemical Kinetic systems. NWChem aims to provide computational chemistry tools that are scalable to large scientific problems and was recently released into open source by the Pacific Northwest National Laboratory [11]. We would like to extend existing work to the chemical kinetics of ion channels and provide more substantive support for automatic performance tuning on GPUs.

V. Results and Remaining Challenges

Details of the performance speedup we have achieved on multicore using Intel's Threaded Building Blocks (TBB) and on a multicore cluster using MPI can be found in this study [12] in which we examined the process of adapting computational biology models to multicore and distributed memory architectures. Fig. 2 demonstrates the performance increase we have achieved using GPU as compared with distributed memory processing using MPI (we also show linear GPU speedup to greater than 100x over sequential CPU that is not shown due to space limitations).

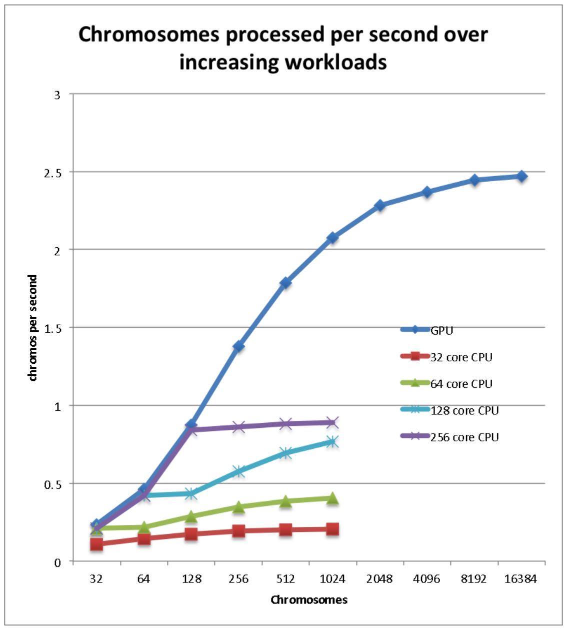

Figure 2. This graph compares the chromosomes processed per second by GPU, 32 core CPU, 64 core CPU, 128 core CPU, and 256 core CPU using MPI. We did not use more than 256 processes on the cluster because only 204 cores are available and at 256 cores the nodes are already oversubscribed.

The speedup we have achieved to date represent a fundamental change in the science that is possible. We have identified a model that has a better fit to experimental data and that more completely describes all key biological characteristics of AMPA receptors that underlie complex neuronal signaling. Fig. 3 demonstrates that 5% to 22% performance increase can be achieved on the GPU without the involvement of the application programmer through better register utilization. Our framework will automatically generate code for optimal block sizes and memory allocation. These are the architectural details so important to optimal performance but that are a barrier to broad use by scientists.

We have advanced computational simulation in kinetic modeling and are well underway with the development of the code generation framework for performance tuning. What remains is to demonstrate that performance is sustained across several models, prepare an in-depth comparison of GPU to CPU, and to integrate automated code verification in the framework. In future research, we would like to determine how generally applicable tools like KPP and NWChem are to the specifics of our chemical kinetics and perhaps provide auto-tuning for performance on the GPU within these systems to take advantage of the broader domain support they already have in place.

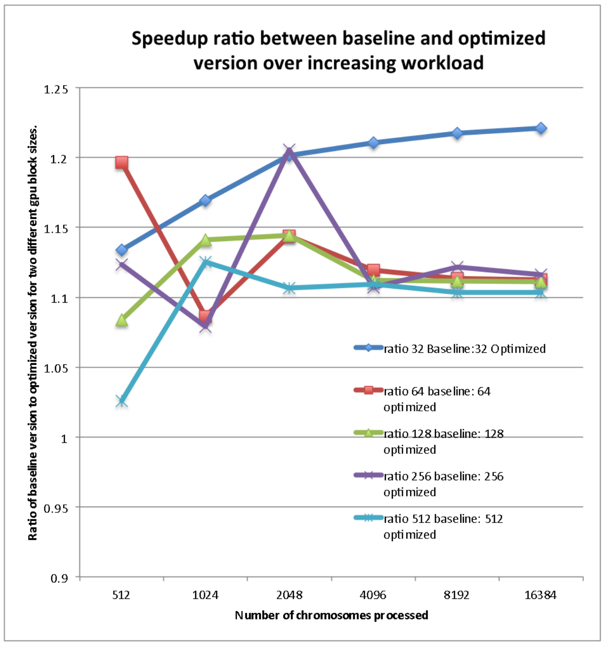

Figure 3. The y-axis shows the speedup between our baseline and an optimized implementation that uses registers more aggressively. Above one on the y-axis means speedup is achieved in the optimized version. The lines are different GPU block sizes (a setting that is automated in our platform but that typically needs to be determined by the programmer with little insight into the consequences of block size on application behavior). The increasing curve of the blue line indicates that with larger workloads memory latency is being hidden more effectively and that latency savings is being magnified over increased data parallelism. The other lines indicate there is a performance sweet spot on the GPU of over 20% that isn't realized with standard compilation.

ACKNOWLEDGMENT

Allison Gehrke is an Achievement Rewards for College Scientists (ARCS) scholar and gratefully acknowledges the support of the ARCS foundation.

REFERENCES

[1] Y. Li, J. Dongarra, and S. Tomov, "A note on auto-tuning gemm for gpus," in *Computational Science ICCS 2009*, ser. Lecture Notes in Computer Science, G. Allen, J. Nabrzyski, E. Seidel, G. van Albada, J. Dongarra, and P. Sloot, Eds. Springer Berlin Heidelberg, 2009, vol. 5544, pp. 884–892.

[2] P. J. Roache, *Verification and Validation in Computational Science and Engineering*. hermosa publishers, 1998.

[3] R. Melnick, M. Siclari, F. Marconi, T. Barber, and A. Verhoff, "An overview of a recent industry effort at cfd code validation," in *26th AIAA Fluid Dynamics Conference*, 2005.

[4] A. Rizzi and J. Vos, "Towards establishing credibility in cfd simulations," in *27th AIAA Fluid Dynamics Conference*, 1996.

[5] D. E. Post, R. P. Kendall, and E. M. Whitney, "Case study of the falcon code project," in *Proceedings of the second international workshop on Software engineering for high performance computing system applications*, ser. SE-HPCS '05. New York, NY, USA: ACM, 2005, pp. 22–26.

[6] A. Robert and J. R. Howe, "How ampa receptor desensitization depends on receptor occupancy," *The Journal of Neuroscience*, vol. 23, no. 3, pp. 847–858, 2003.

[7] J. C. Linford, J. Michalakes, M. Vachharajani, and A. Sandu, "Multi-core acceleration of chemical kinetics for simulation and prediction," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, ser. SC '09. New York, NY, USA: ACM, 2009, pp. 7:1–7:11.

[8] E. Jakobsson, R. J. Mashl, and T.-T. Tseng, "Investigating ion channels using computational methods," in *Peptide-Lipid Interactions*, ser. Current Topics in Membranes, T. J. M. Sidney A. Simon, Ed. Academic Press, 2002, vol. 52, pp. 255 – 273.

[9] K. Rennie, M. Streeter, T. Benke, and A. Moritz, "Modeling channel properties in vestibular calyx terminals," *Biomedical sciences instrumentation*, vol. 41, pp. 358–363, 2005.

[10] V. Damian, A. Sandu, M. Damian, F. Potra, and G. R. Carmichael, "The kinetic preprocessor kpp-a software environment for solving chemical kinetics," *Computers And Chemical Engineering*, vol. 26, no. 11, pp. 1567–1579, 2002.

[11] M.Valiev, E. Bylaska, N.Govind, K.Kowalski, T. Straatsma, H. van Dam, D. Wang, J. Nieplocha, E. Apra, T. Windus, and W. de Jong, "Nwchem: a comprehensive and scalable open-source solution for large scale molecular simulations," *Computer Physics Communications*, vol. 181, pp. 1477–1489, 2010.

[12] A. Gehrke, K. Rennie, T. Benke, D. A. Connors, and I. Ra, "Modeling ion channel kinetics with hpc," *High Performance Computing and Communications, 10th IEEE International Conference on*, vol. 0, pp. 562–567, 2010.