IEICE Proceeding Series

Coarse grain parallelization and acceleration of biochemical ODE simulation using GPGPU

Kazushige Nakamura, Kei Sumiyoshi, Noriko Hiroi, Akira Funahashi

Vol. 1 pp. 29-32
Publication Date: 2014/03/17
Online ISSN: 2188-5079

2012 International Symposium on Nonlinear Theory and its Applications
NOLTA2012, Palma, Majorca, Spain, October 22-26, 2012

Coarse grain parallelization and acceleration of biochemical ODE simulation using GPGPU

Kazushige Nakamura† , Kei Sumiyoshi†, Noriko Hiroi†, Akira Funahashi†

†Graduate School of Science and Technology, Keio University
3141 Hiyoshi, Kouhoku-ku, Yokohama, 223-8522 JAPAN

Abstract—We have accelerated the simulation of biochemical ODE model described in SBML(Systems Biology Markup Language), by using the parallel processing approach on GPU. Compared with the implementation on CPU, our simulator have accelerated about 12 times faster in a single-precision number, and 10 times faster in a double- precision. In this research, we have implemented the sim- ulator which have the function to read models dynamically from SBML files, and simulates the model on GPU. Sev- eral existing works have done which numerically solves ODE on GPU. However, those simulators could not read models dynamically without generating a code for every model. Our work achieved better portability than previ- ous researches by reading models dynamically from SBML files, without a re-compiling of the codes. We implemented the solver by the classical Runge-kutta method, which has known to be a basic factor of other solvers, therefore we can develop advanced solvers in the future based on our current simulator.

1. Introduction

In System Biology, a mathematical modeling is a useful method for analyzing biochemical networks. Some of the mathematical models are written by a boolean network or a stochastic models, but most of the models are written by Ordinary Differential Equation (ODE)[1]. Systems Biology Markup Language(SBML) is the standard for describing biochemical models which is written by XML, and several biochamical models are already written by SBML[2]. There are many software tools, which can read the models written by SBML, such as CellDesigner[3] and Copasi[4]. When simulating an ODE, the result would be identical depending on the initial condition. However, parameter fittings are necessary to gain a feedback to the original experiment. In addition, we can understand the model by analyzing a model, such as sensitivity analysis and bifurcation analysis. When doing a parameter fitting or an analysis, we must run multiple simulations to obtain results from various parameter sets, which leads to find the most suitable parameters. Therefore, a parameter fitting or an analysis of a model will result to a large computational cost.

To solve this problem, our goal is to accelerate the simulator by using General Purpose Computation on Graphics Processing Unit (GPGPU) to provide better environment for researchers by individual level. GPGPU uses a device called GPU, which is used to draw graphical elements in a computer. GPU has better cost performance compared to other parallel computational device, and it is easy to introduce in a modern research lab[5].

Integration using GPU has been done by past researches, and it has shown a good result[6]. In addition, there are a research specific to the biochemical reaction, and it calculates ODE by generating a code[7]. But the simulator which generates a code for each models is difficult in general-use for ODE solver. In our research, we implemented a simulator which can read an SBML file dynamically, so the generation of the code is not required.

In this paper, we introduce details about a GPU computing in chapter 2, implementation about solving a ODE on the GPU in chapter 3, and profiling of the simulator and discussion in chapter 4.

2. GPU Computation

Graphics Processing Unit (GPU) is used to draw graphical elements in a computer. General Purpose computation on GPU (GPGPU) uses GPU as general computational tool other than drawing a graphic, such as hydrodynamic calculations or vector analysis, which requires a large number of simple calculations. A GPU board, which is used in GPGPU, is sold for modern consumer market, therefore a GPU costs less than other high-performance calculation system. A GPU costs about few hundred dollars in a retail store, which is cheaper than parallel computational device such as FPGA and PC cluster. The below listed are the advantages of a GPU, as a computational device.

- Higher computational performance than a single CPU
- High growing of a performance throughout the decade
- Costs less than other parallel computational device
- Requires Less space
- Consume less energy per calculation

Typically, OpenCL and Compute Unified DeviceArchitecture (CUDA) are used for GPGPU implementation. CUDA is provided form NVIDIA Corp. and by using CUDA, we can develop the program based on C. We have

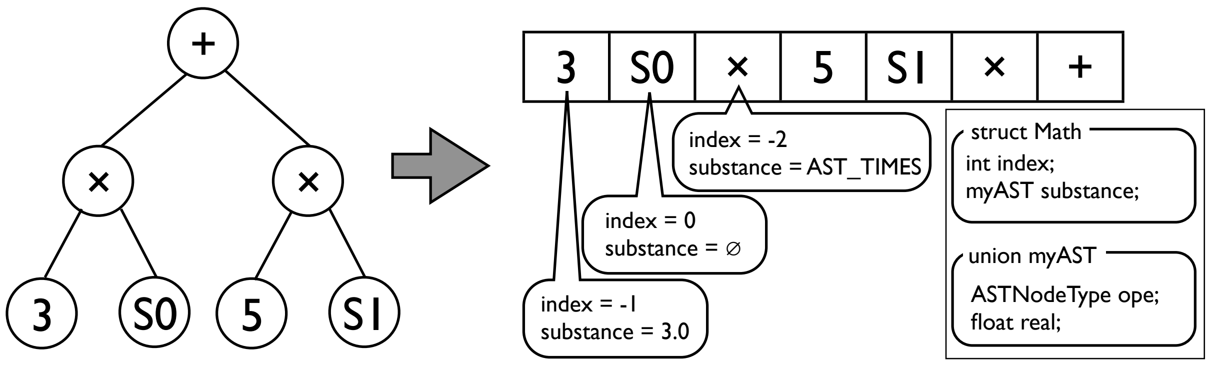

Figure 1: Searching an index of "Math structure" determines whether it is a constant, a real number, or arithmetic operators. If it is a real number of an arithmetic operator, gets corresponding information from shared "substance".

chosen to use CUDA for implementation, because it is designed specific for GPU computing, and past researches are using CUDA for implementation.

2.1. GPU architecture

Understanding GPU architecture is necesary for GPU computing. GPUs are produced by NVIDIA Corp. and AMD Corp., but using CUDA is limited to NVIDIA GPUs. In this chapter, we introduce an architecture about NVIDIA GPUs.

An architecture of GPU is different by their version throughout the generation, but the "Compute Capability 1.x" version has a same architecture between each other. A high performance of GPU has achieved by many computational unit called "CUDA Core". On the GPU chip, there are several SM (Streaming multi processer), and inside each SM, there are computational unit called "CUDA Core". 32 CUDA Core are loaded per one SM (Before the recent version, there are 8 Streaming Processor per SM). When GPU calculates in parallel, each multiprocessor creates, manages, schedules, and executes threads in groups of 32 parallel threads called warps in GPU computing. Therefore, same type of the calculation must be done by same warp threads to make efficient implementation.

2.2. Memory management

In order to calculate, GPU copies the data from a main memory to their device memory. A GPU has six different types of device memory, and they are Register, Local Memory, Shared Memory, Global Memory, Constant Memory, and Texture Memory. In this chapter, we focus on Global Memory and the Constant Memory.

Global Memory has a large size for read and writing, but the speed of its access is slow. Currently, GPU executes a half-warp (16 threads) per one access. The speed of access from GPU to non-GPU memory is about hundred-times slower than the access from GPU to GPU memory, such as Global Memory, therefore we must concern about the time required for memory access. An access to Global Memory is done by 32, 54, or 128 bytes unit, which is aligned by 32, 54, or 128 bytes boundary. When accessing to the memory, GPU adds up the memory address consisted by half-warp (16 threads) which can execute at the same time, and transfer all memories simultaneous by. This access can be lead to the less access to the memory, which is called "coalescing", and GPU typically uses this memory access to gets an efficient performance. Our implementation saves amounts, reaction constants, and time course to global memory, and we optimized a structure of the data so every access would be coalesced.

Constant Memory is a "read-only memory", but this memory can use a cache. Constant Memory can not allocate a memory dynamically, and the capable memory size is low as 64k bytes. In contrast, when multiple threads are accessing on the same memory address, the cache becomes active and efficient access will be archieved. In our implementation, we have moved the read-only data to Constant Memory, such as stoichiometry matrix and differential equation.

3. Implementation

In our research, we simulate a model by reading an SBML model and solves an ODE. Reaction equation and stoichiometry can obtained from an SBML file, but the problem is how to handle a differential equation in GPU. In SBML, we can get information from Abstract Syntax Tree (AST), but GPU does not have an ability to use a recursive function to scan the tree. To solve this problem, we converted the tree structre to the array structure by Reverse Polish Notation (RPN) and transfered to GPU. In addition, we want to use Constant Memory for these data, so the size of the data is prefered to be smallest as possible.

			full	ODE	SMat	store	memory
mapk	1,024	time(sec.)	9.861	5.115	1.81	0.05	2.812
		rate(%)	100	51.9	18.6	0.5	28.5
	2,048	time(sec.)	10.613	5.128	1.953	0.059	3.516
		rate(%)	100	48.3	18.4	0.6	33.1
CellCycle	1,024	time(sec.)	14.349	9.415	2.356	0.046	2.716
		rate(%)	100	65.6	16.4	0.3	18.9
	2,048	time(sec.)	15.141	9.599	2.739	0.031	3.222
		rate(%)	100	63.4	18.1	0.2	21.3

Table 1: Execution time in each part of the simulation, compared to the total execution time. Execution time depends on the size of the model and does not depend on the number of simulation.

In our implementation, we stored the equation written as an AST node the node to the following data structure represented in Figure1. When describing a real number by float, the size would be only 8 bytes per one "Math" structure, so it can be easly handled on the Constanty Memory. In the current implementation, few functions such as "Event" and "delay" in SBML is not available, but when consider about delay function, we can still keep up with large models which has hundreds of reactions and species.

4. Result

We used a benchmark as MAPK model[8] in BioModels[9], which has 10 reactions and 8 species. We implemented a solver by two ways, which is a constant step with a classical Runge-Kutta method, and the other is a changeable step with a low storage RK[10]. In the benchmark, we used a method of a constant step with a classical Runge-Kutta method, because it is easier to see the computational performance in GPU. We measured the simulation time in $T = 2,000$ with a step distance of $h = 0.1$, which have a total of 20,000 steps, and calculated a mean of 10 realizations. On the calculation, we used CPU of Intel Core i7 2.8GHz, and the GPU is Tesla C2070. We measured the time from starting an integration calculation to end of the calculation. GPU has Global Memory of 5G bytes, with Compute Capability of 2.0. On the GPU, we also included the time of data transfer of time course to the measurement. The time of initializing, such as reading an SBML file and converting differetial eauations to RPN format are significantly short so we ignored these initializations to the measurement.

Change of a calculation time by $1 \sim 2,048$ realization has shown in Figure 2. The calculation time of CPU grows linear by increasing the realization, but the calculation time of GPU does not change. The slight increase of a time would be leads to the allocation and copying of memories. When running 2,048 realizations, GPU runs 12 times faster in a single precise, and 10 times faster in a double precise than CPU. In this benchmark, we recorded time courses in 3G bytes of memories, so the allocating and copying of the memory took about 30% of the total time, which is not

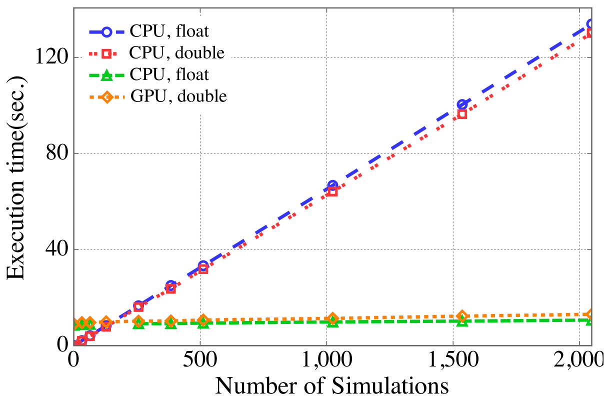

Figure 2: Simulation time on MAPK model. We have achieved 12 times faster in single precise, and 10 times faster in double precise compared to CPU.

part of the calculation. From this result, this simulator is said to be very useful for flexible calculation and analyzing purpose which does not records the time courses.

In addition, there are no big effect on the calculation speed between single and double precise. We must pay an attention to the decreace in occupancy and increase in memory transfer size, according to the many usage of Register. We used a recent GPU which has more Register for double precise than GPU such as older than Fermi generation, so it seems to be no significant change between single and double precise calculation, unless we satisfy the coalescing rule. When implementing an Implicit Method for calculation, or implementing "Algebraic Rule" in SBML, the optimizing implementation for a double precise will be necessary, but from our result, it can say that better calculating efficiency could be archived than CPU.

Execution time of 1,024 and 2,048 realizations is shown in Table 1. In Table 1 "ODE" is the part of solving a differential equation and calculating each reaction. "SMat (Stoichiometry Matrix)" is the part of calculating flux of species by stoichiometry and each reaction. "Store" is the part of storing the time course to Global Memory. "Memory" is the other part such as allocating or copying the memory. To show the contrast, the execution time of CellCycleModel[11] is shown, which is listed in number 181 of BioModels, which has 18 reactions and 6 species. The number of ASTNode of MAPK is 106, and CellCycle model is 156 respectively. The execusion time of ODE is proportional to the number of ASTNode, therefore simulating a larger model would be a bottle-neck of the total execution time. In contrast, the execution time in SMat, or stoichimetry matrix is depending on the species and reaction number, so the execution time would be larger, but the calculation of ODE is the most part of the whole execution time. Using list structure instead of a matrix can use Constant Memory even on the large model.

To solve the problem of ODE caltulation time, using the "fine-grained" parallelization method would be useful. Making many threads as same as the number of reaction (when the reaction number exceeds the block's warp number, it becomes the block's maximum warp number) could make each warp to calculate different reaction, leads to shorter calculation of ODE. In addition, SMat calculation can also parallelized, so additional acceleration of the simulation is expected.

5. Conclusion

In this research, we have implemented a simulator which executed an simulation about 10 times faster than CPU. Based on this work, we can implement Implicit Method and other functions optimized to SBML, to execute parameter fittings and parameter scans. The bottle-neck of the calculation can be solved by implementing the fine-grained parallelization method, and this implementation can lead to more acceleration on the simulation.

References

[1] K. Hübner, S. Sahle, and U. Kummer. Applications and trends in systems biology in biochemistry. *FEBS Journal*, Vol. 278, No. 16, pp. 2767–2857, 2011.

[2] M. Hucka, A. Finney, B.J. Bornstein, S.M. Keating, B.E. Shapiro, J. Matthews, B.L. Kovitz, M.J. Schilstra, A. Funahashi, Doyle J.C., and H. Kitano. Evolving a lingua franca and associated software infrastructure for computational systems biology: t he Systems Biology Markup Language (SBML) project. *IEE Systems Biology*, Vol. 1, No. 1, pp. 41–53, September 2004.

[3] A. Funahashi, Y. Matsuoka, A. Jouraku, M. Morohashi, N. Kikuchi, and H. Kitano. CellDesigner 3.5: A versatile modeling tool for biochemical networks. *Proceedings of the IEEE, Special Issue: Computational Systems Biology*, Vol. 96, No. 8, pp. 1254–1265, August 2008.

[4] S. Hoops, S. Sahle, R. Gauges, C. Lee, J. Pahle, N. Simus, M. Singhal, L. Xu, P. Mendes, and U. Kummer. COPASI–a COmplex PAthway SImulator. *Bioinformatics*, Vol. 22, No. 24, p. 3067, 2006.

[5] *NVIDIA CUDA Compute Unified Device Architecture Programming Guide Version 4.1.* `http://developer.nvidia.com/nvidia-gpu-computing-documentation`.

[6] L. Murray. Gpu acceleration of runge-kutta integrators. *Parallel and Distributed Systems, IEEE Transactions on*, No. 99, pp. 1–1, 2011.

[7] J. Ackermann, P. Baecher, T. Franzel, M. Goesele, K. Hamacher, et al. Massively-parallel simulation of biochemical systems. *Proceedings of Massively Parallel Computational Biology on GPUs*, 2009.

[8] B.N. Kholodenko. Negative feedback and ultrasensitivity can bring about oscillations in the mitogen-activated protein kinase cascades. *European Journal of Biochemistry*, Vol. 267, No. 6, pp. 1583–1588, 2000.

[9] Nicolas Le Novère, Benjamin Bornstein, Alexander Broicher, Mélanie Courtot, Marco Donizelli, Harish Dharuri, Lu Li, Herbert Sauro, Maria Schilstra, Bruce Shapiro, Jacky L. Snoep, and Michael Hucka. BioModels Database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems. *Nucleic Acids Research*, Vol. 34, No. Database issue, pp. D689–D691, Jan 2006.

[10] C.A. Kennedy, M.H. Carpenter, and R.M. Lewis. Low-storage, explicit runge–kutta schemes for the compressible navier–stokes equations. *Applied numerical mathematics*, Vol. 35, No. 3, pp. 177–219, 2000.

[11] K. Sriram, G. Bernot, F. Kepes, et al. A minimal mathematical model combining several regulatory cycles from the budding yeast cell cycle. *IET systems biology*, Vol. 1, No. 6, pp. 326–341, 2007.