GPU Accelerated Stochastic Simulation

David D. Jenkins, Gregory D. Peterson
Department of Electrical Engineering and Computer Science
University of Tennessee
Knoxville, Tennessee, USA
Email: {ddj,gdp}@utk.edu

***Abstract*—Through computational methods, biologists are able learn more about molecular biology by building accurate models. These models represent and predict the reactions among species populations within a system. One popular method to develop predictive models is to use a stochastic, Monte Carlo method developed by Gillespie called the stochastic simulation algorithm (SSA). Since this algorithm is based on stochastic principles, large numbers of simulations are needed to provide quality statistical models of the species and their interactions, giving way to long runtimes for large systems. In this paper, we provide an implementation of SSA onto NIVIDA graphics processing units using CUDA to parallelize ensembles of simulations. With this implementation we are able to see up to 41.9x speedup over the best-known serial implementations.**

***Index Terms*—Gillespie; SSA; GPU; CUDA;**

I. Introduction

With the advancement of systems biology comes the need to observe and understand the reactions among molecular species. One way to do this is to generate predictive models of the time evolution of spatially homogeneous mixtures of chemically reacting molecules using deterministic or stochastic approaches. A scientist can use deterministic approaches involving ordinary differential equations (ODEs) to model a system; however, these approaches are inaccurate for modeling small species populations because they do not consider statistical fluctuations and correlations inherent in chemical reactions. The use of ODEs are not able to correctly model nonlinear systems. Also, ODEs have no sense of discrete values for species populations. Therefore, we employ the stochastic approach using the stochastic simulation algorithm (SSA).

Dan Gillespie first developed the SSA approach in 1976 to help combat the previously mentioned issues with traditional deterministic approaches [1], [2]. Using Monte Carlo methods, SSA is used to exactly predict the execution time and identity of each reaction. One run of SSA gives one possible solution to the Chemical Master Equation (CME) in equation 1 [3], [4], that describes all species populations for any given time step. Due to the stochastic nature of the algorithm, many simulations are needed to produce statistically accurate models of these systems. In the serial case, this becomes extremely time consuming, especially for large systems.

$$\frac{\partial}{\partial t}P(x,t|X(0),t_0) = \sum_{j=0}^{M-1}[\alpha_j(x-\nu_j)P(x-\nu_j,t|X(0),t_0) - \alpha_j(x)P(x,t|X(0),t_0)] \quad (1)$$

Previous research examined the ability to accelerate individual simulations by using different calculation methods [1], [2], [4], [6]–[10]. Despite these computational improvements, performing many SSA runs is still a time consuming task. Because of this, we developed an implementation on NVIDIA GPUs using CUDA [12] to perform many ensembles of simulations simultaneously to speedup the overall runtime.

II. Methodology

To give a solution to the CME, the SSA takes an initial set of species populations and reactions, determines the next reaction time by random sampling of a distribution, calculates the likelihood of each reaction occurring in the next time step and it selects one based on their relative likelihood, and applies the selected reaction to the species populations (refer to figure 1). This process is repeated until a specified end reaction time is reached or until a specified number of reactions are executed. Although, there are a number of methods designed to accelerate a single simulation, we decided to focus on one of the original methods, the Direct Method [1], [2]. This method reduces the amount of random numbers and expensive floating point divisions. It also eliminates the need for maintaining a dependency graph for reactions thus reducing the amount of memory usage.

As previously mentioned, the SSA calculates the species populations, $X(t) = X_1(t), X_2(t), ..., X_N(t)$ where N is the number of species, for each time step of the system. Within each time step, a reaction, $R = R_1, R_2, ..., R_M$ where M is the number of reactions, is chosen based on each reaction's propensity (likelihood), α_j where j is the j^{th} reaction, to occur.

First, the propensity of each reaction is calculated based on the populations and coefficients, l_i where i is the species index, of the reaction's reactants, r_j where j is the set of reactants of the j^{th} reaction, as well as the reaction's stochastic rate constant, k_j. The propensity can be calculated as follows [1]:

$$\alpha_j = k_j * \prod_{i \in r_j} \binom{i}{l_i} \quad (2)$$

Second, the time step that the next reaction that will occur is calculated by selecting an exponentially distributed random number and dividing it by the sum of the propensities, α as shown in equation 3 where URN is a uniform random number.

$$\tau_{next} = \frac{-ln(URN)}{\alpha} \quad (3)$$

1. Initialization : Read the model file and initialize all data structures.
2. Propensity Calculation : Calculate the propensity function for each reaction.
3. Reaction Time Generation : Generate random sample of the occurrence time of the next reaction.
4. Reaction Selection : Select which reaction will occur next.
5. Reaction Execution : Update the current simulation time and species variables to reflect the execution of the selected reaction.
6. Termination : If the simulation time < t_{end}, go to step 2.

Fig. 1. Steps to execute the Stochastic Simulation Algorithm

Third, the next reaction to occur is selected by multiplying a uniformly distributed random number by the total sum of the propensities. This number is then incrementally subtracted by the propensity of each reaction until the value reaches less than or equal to 0. The index of the reaction whose propensity causes the value to reach this bound is the index of the next reaction.

Finally, the selected reaction is executed by updating each of the species populations involved in the reaction as described by equation 4.

$$X(t) = X(t-1) - r_{selectedReaction} + p_{selectedReaction} \quad (4)$$

These steps can be summarized in the algorithm pseudo code shown in algorithm 1.

Algorithm 1 Pseudocode for the Direct Method. Courtesy of [5]

```
CurrentTime = 0.0                                          1. Initialization
X[1...M] = Initial Species Populations
R[1...N] = Reactions
TotalPropensity = 0.0
                                                           2. Propensity Calculation
for I = 1...N do
   Prop[I] = CalcPropensity(S,R[I])
   TotalPropensity += Prop[I]
end for
T = -ln(rand())/TotalPropensity                            3. Reaction Time Generation
Selector = TotalPropensity * rand()                        4. Reaction Selection
for I = 1...N do
   Selector - Prop[I]
   if Selector ≤ 0 then
      SelRxn = I
      break
   end if
end for
X = X - R[SelRxn].reactants + R[SelRxn].products           5.Reaction Execution
CurrentTime += T;
                                                           6. Termination
if CurrentTime < EndTime then
   GOTO Propensity Calculation
end if
```

III. Graphics Processing Unit Implementation

General-purpose computation on graphics processing units (GPGPUs) has grown to be an enormous research area for general purpose computing. This can be attributed to relative ease of programming compared to other accelerator technologies (e.g., FPGAs), efficient parallel pipelines, and efficient floating point performance. For this work, we explored the use of both an NVIDIA Tesla c1060 (4GB RAM, 16KB shared memory) and an NVIDIA GeForce GTX 480 (1.5GB RAM, 49KB shared memory). The Tesla c1060 is the previous generation of GPUs whereas the GeForce GTX 480 is the current generation Fermi architecture containing 480 processing cores. Since we are using NVIDIA GPUs, we are also using CUDA (compute unified device architecture) as our programming language of choice.

Due to the iterative nature of SSA, it is extremely difficult to parallelize a single simulation on the GPU. However, we are able to exploit the need to run many simulations by performing ensembles of simulations concurrently. To do this, we delegated each thread to do a single simulation for a set number of reactions.

Unlike previous implementations [13], this work contains only one kernel that performs the simulations. Random numbers are generated with a linear feedback shift register to help reduce the about of kernels needed for execution. This simple implementation was used because there are currently no random number generators for the GPU available that give numbers on a thread-by-thread basis.

Since this application contains copious amounts memory accesses, it was necessary to coalesce memory reads and writes as well as limit the use of global memory. To do this, we allocate and arrange memory so that each thread with in a block is reading/writing in consecutive memory locations. Also, to combat the usage of global memory, we loaded information that is repetitively used into registers such as random number generator seeds, propensity totals, and temporary intermediate values.

Our final optimization was to limit the amount of divergent branches. We simply replace some conditionals with logic operations. This causes extra computations; however, it proved beneficial for our implementation.

IV. Results

To examine the performance, we run our program against the fastest known serial implementation [14]. We use a few models varying in sizes described in table I. All the model files came from [6].

Both implementations were written in C and/or CUDA and compile with the Intel C complier version 11.1 and NVIDIA's CUDA version 3.0. Each of the codes was compiled with the highest optimization flags. The serial implementation was run on a 2.93GHz Intel Core i7 with 24GB RAM.

For each implementation, multiple numbers of simulations are run to get a trend of the speedup compared to the serial version. These numbers of simulations vary by powers of two from 1024 up to 32768. For each increment in the number of simulations, we complete 10 runs to get an accurate average simulation runtime.

Figure 2 shows the speedup of our application over the serial implementation using the Tesla c1060. We achieve a maximum speedup of 8.5x over the serial implementation with the DIMER model, 8.5x with the HSR model, and 5.9x with the QS8 model. This difference in performance can be attributed to the larger number of memory accesses for the larger models. As expected, as the number of simulations doubles, so does the runtime for both implementations thus giving us a roughly linear speedup. Saying this, with a smaller

TABLE I
MODELS USED FOR COMPARISONS

Model	# Species	# Reactions
DIMER	8	13
HSR	28	61
QS8	122	201

number of simulations, we see less of a speedup due to transfer costs of data to/from the GPU. With larger numbers of simulations, these transfer costs are masked by the runtime of the application.

Figure 3 shows the speedup of our application over the serial implementation using the GeForce GTX480. As can be seen from the figure, we achieve a maximum speedup of 42x over the serial implementation with the DIMER model, 25.7x with the HSR model, and 16.5x with the QS8 model. The differences in performance among models are the same as described above. One thing to note here is that we did not change the code between the two cards. We simply recompiled the same code and ran the on each of the cards. Doing this gave us an additional 4.9x speedup for the DIMER model, 3.0x for the HSR model, and 2.9x for the QS8 model.

V. Conclusion

In this paper, we examined the use of GPUs with CUDA to accelerate the stochastic simulation algorithm. Due to data dependencies, each simulation is not targeted for acceleration, but rather the parallel simulation of an ensemble of simulations to provide good statistical analysis. We find that the GPU provides excellent acceleration for the SSA algorithm while maintaining the exact statistical properties of the chemical master equation. With the recent release of the Fermi architecture GPUs, this acceleration is further increased without any code changes. Future work would be to port this to OpenCL to explore the use of other types of GPUs such as the ATI 5870.

Acknowledgment

This work is partially supported by the National Science Foundation, grant NSF CHE-0625598.

References

[1] D. T. Gillespie, "A General Method for Numerically Simulating the Stochastic Time Evolution of Coupled Chemical Reactions", *J. Comp. Phys.*, vol. 22, no. 4, pp. 403-434, December 1976.
[2] D. T. Gillespie, "Exact Stochastic Simulation of Coupled Chemical Reactions", *J. Phys. Chem.*, vol. 81, no. 25, pp. 2340- 2361, 1977.
[3] D. T. Gillespie, "A rigorous derivation of the chemical master equation", *Physica A: Statistical and Theoretical Physics*, vol. 188, no. 1-3, pp. 404-425, September 1992.
[4] D. T. Gillespie, "Approximate Accelerated Stochastic Simulation of Chemically Reacting Systems", *J. Chem. Phys.*, vol. 115, pp. 1716-1733, 2001.
[5] J. M. McCollum, "Accelerating Exact Stochastic Simulation", MS thesis, University of Tennessee. 2004.
[6] J. M. McCollum, "Accelerating Exact Stochastic Simulation of Biochemical Systems", PhD dissertation, Electrical and Computer Engineering, University of Tennessee. 2006.
[7] J. M. McCollum, et. al., "The sorting direct method for stochastic simulation of biochemical systems with varying reaction execution behavior", *Computational Biology and Chemistry*, vol. 30, No. 1, pp. 39-49, February 2006.
[8] M. A. Gibson and J. Bruck, "An Efficient Algorithm for Generating Trajectories of Stochastic Gene Regulation Reactions," California Institute of Technology, Pasadena, CA, Technical Report 1998.
[9] M. A. Gibson and J. Bruck, "Efficient Exact Stochastic Simulation of Chemical Systems with Many Species and Many Channels," *Journal of Physical Chemistry*, vol. 104, p. 23, March 2000.
[10] H. Li and L. R. Petzold, "Logarithmic Direct Method for Discrete Stochastic Simulation of Chemically Reacting Systems," Technical Report, p. 11, 27 July 2006 2006.
[11] H. Li and L. Petzold, "Efficient Parallelization of Stochastic Simulation Algorithm for Chemically Reacting Systems on the Graphics Processing Unit", Technical report, Dept. Computer Science, University of California, Santa Barbara, 2007.
[12] NVIDIA Corporation, "NVIDIA CUDA Compute Unified Device Architecture Programming Guide", February 2010 [Online], http://developer.download.nvidia.com.
[13] D. Jenkins and G. D. Peterson, "Accelerating the Stochastic Simulation Algorithm," presented at the Symposium on Application Accelerators in High-Performance Computing, Urbana, IL, 2009.
[14] D. D. Jenkins, "Accelerating the Stochastic Simulation Algorithm Using Emerging Architectures." Master's Thesis, University of Tennessee, 2009. http://trace.tennessee.edu/utk_gradthes/533

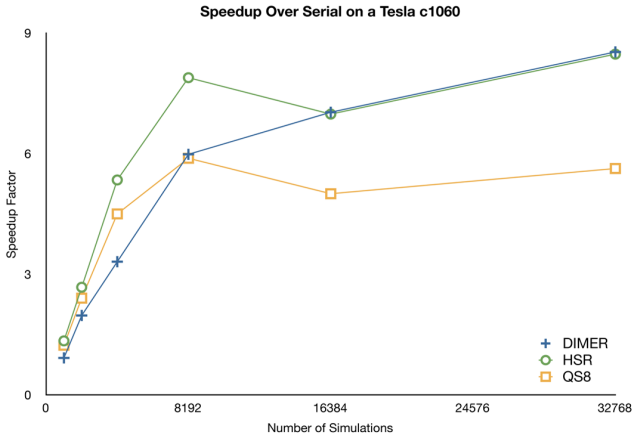

Fig. 2. Speedup of the NVIDIA Tesla c1060 over the serial implementation.

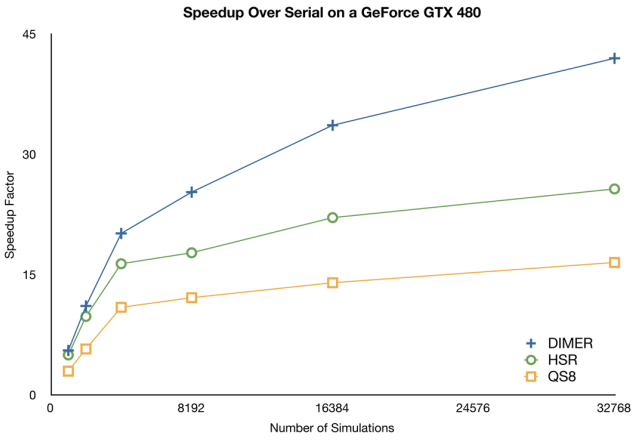

Fig. 3. Speedup of the NVIDIA GeForce GTX 480 over the serial implementation.