Parallel studies for chemically reacting systems

Yushu Yang and Muruhan Rathinam *

August 4, 2009

Abstract. Parallel computing code can be applied to solve chemically reacting systems. In a well stirred chemical system, the number of molecules for each species can be solved by implicit tau method, and the histogram of the method is used to compare with the exact simulation called SSA. In this report, the parallel code will be implemented in both SSA and implicit tau, with the discussion that how random number generator will be applied in the parallel code from large number of sample simulations. Moreover, the performance study of the parallelism and some statistical analysis will be provided.

1 Introduction

The general problem to be solved here is the exact approximation and numerical solution of the stochastic chemical system by stochastic simulation algorithm (SSA) and implicit tau method. Since both SSA and implicit tau involve with generating random variables, the result will be analyzed through generating large samples of simulations. In parallel computing, this can be achieved by distributing all the simulations into different processes and then collect the data into one process.

In this report, section 2 describes the background of the chemical reaction systems and random number generator. Section 3 discusses how the parallel implementation applies to the chemical systems. Section 4 uses the time performance, numerical results and plots to illustrate the advantages of parallel code to solve some numerical problems by using random numbers or variables.

2 The stochastic reaction system background

2.1 Stochastic chemical reaction systems and SSA

Stochastic chemical reaction systems involved with small number of molecules have a dynamic behavior that is discrete and stochastic. However, for large number molecules of

*Department of Mathematics and Statistics, University of Maryland, Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250. yushu1,muruhan@umbc.edu

species, the number of species can be modeled by some deterministic differential equation. The following part will introduce the stochastic and deterministic behaviors of the chemically reacting systems [1].

The standard well-stirred chemical model is that there are well-stirred mixture of N molecular species $\{S_1, \ldots, S_N\}$ interacting through M chemical reaction channels $\{R_1, \ldots, R_M\}$. Note that the state space of the system is denoted by $(X_1(t), \ldots, X_N(t))$, where $X_i(t)$ is the number of molecules S_i at time t. For each $j = 1, \ldots, M$, $a_j(x)\tau + o(\tau)$ is the probability, given $X(t) = x$, that reaction R_j will occur in $(t, t+\tau]$, where $a_j(x)$ is the *propensity function*. Vector $\nu_j, j = 1, \ldots, M$ is the *stoichiometric vector*, whose ith component ν_{ji} is the change in the number of S_i molecules produced by one R_j reaction.

$X(t)$ can be simulated exactly by the stochastic simulation algorithm (SSA) [2]. SSA is based on the *next-reaction density function* $p(\tau, j|x, t)$ which is defined as the probability, given $X(t) = x$, that the next reaction in the system will occur in the infinitesimal time interval $(t + \tau, t + \tau + dt]$ and will be a R_j reaction. It follows that

$$p(\tau, j|x, t) = a_j(x)\exp[-a_0(x)\tau] \ (\tau \geq 0; j = 1, \ldots, M), \tag{2.1}$$

where $a_0(x) = \sum_{j=1}^{M} a_j(x)$. The SSA generates τ and j according to (2.1) and then advances the system according to

$$X(t + \tau) = X(t) + \nu_j.$$

2.2 Implicit tau method

If the number of molecules is very large, the randomness in the trajectory of $X(t)$ is not noticeable. In this case, the trajectory takes on the character of continues and deterministic process and can be described by the following reaction rate equation (RRE) [3].

$$\dot{\bar{X}}(t) = \sum_{j=1}^{M} \nu_j \, a_j(\bar{X}(t)), \tag{2.2}$$

with initial condition $\bar{X}(0) = x_0$.

Since SSA simulates one reaction each time, it is very computationally expensive. The tau-leaping methods [4, 5] are proposed to accelerate the chemical reaction simulation. They proceed as follows. First a time step τ is chosen. Define $R_j(x, \tau)$ to be the number of times, given $X(t) = x$, that jth reaction channel will fire in the time interval $(t, t+\tau], (j = 1, \ldots, M)$. Then

$$X(t + \tau) = x + \sum_{j=1}^{M} \nu_j R_j(x, \tau). \tag{2.3}$$

In general, the distribution of $R_j(x, \tau)$ is not known. In a tau leap method, an approximation to $R_j(x, \tau)$ is computed. The formula of the implicit tau method, inspired by the implicit Euler method, is given by

$$X^{(it)}(t+\tau) = x + \sum_{j=1}^{M} \nu_j \{P_j(a_j(x)\tau) - a_j(x)\tau + a_j(X^{(it)}(t+\tau))\tau\}. \tag{2.4}$$

The method approximates $R_j(x,\tau)$ by $P_j(a_j(x)\tau)$, where $P_j = P_j(a_j(x)\tau)$ are statistically independent Poisson random variables.

Newton's method is applied to solve (2.4). Note that $X^{(it)}(t+\tau)$ is not an integer vector any more. For the final record, we should round the values to the nearest integer.

2.3 Random Number Generator

A simulation of any system with random components requires a method for generating numbers that are random. A random number is a random variable which is uniformly distributed over [0,1]. Random variables from all other distributions (e.g., normal, gamma, binomial, Poisson) can be obtained by transforming random numbers. It produces stream of random numbers that appear to be independent identically distributed. All practical random number generators produce only a finite sequence which is then repeated. Hence, these random numbers are also called pseudo-random numbers (PRN).

One can use theoretical and empirical tests to see if a PRN generator passes statistical tests or not. Generally, a good random number generator should have following properties: it should be distributed uniformly on [0,1] and independent of each other; it should be able to reproduce a particular stream of random numbers; the cycle length takes long before numbers start to repeat; the speed should be fast, and requires little storage.

A seed should be initialized to some distinctive value using random number generator. For every different seed value used in a call, the pseudo-random number generator can be expected to generate a different succession of results in the subsequent calls. Two different initializations with the same seed, instructs the pseudo-random generator to generate the same succession of results for the subsequent calls [6].

3 Parallelism of SSA and Implicit tau method

The purpose of studying chemically reacting system with N species interacting through M reactions is to compare the number of molecules for each species after a given time T. Let $X^{(ssa)}(T)$ and $X^{(it)}(T)$ represent the number of molecules of the species for SSA and implicit tau at time T.

As stated before, (2.1) and (2.4) give the numerical algorithm how to get the number of molecules after time T. Since such process involves with generating the random variables, the results should be based on generating a large sample of simulations R, and compare the histogram between the two methods used. Statistically, larger R gives more accurate results. There idea of parallelism of SSA and implicit tau is that we evenly distribute the number of simulations R into np processes, with each process generates $l_R = R/np$ samples. After the process of the simulation, all the data can be obtained by `MPI_Gather`, which collects the data from each process and stores the data in process rank 0.

Total wall clock time in seconds for SSA and implicit tau							
p=1	p=2	p=4	p=8	p=16	p=32	p=64	p=128
604	300	148	76	39	20	11.0	5.8

Table 1: Time table by number of processes used with 4 processes per node, except for $p = 1$ using 1 process per node, and $p = 2$ using 2 processes per node for both SSA and implicit tau.

Another point for such numerical method with random variables is to find a way of setting up the random number generator on the parallel processors. Here I use the random number generator from C library. r=`rand()` returns an integer value between 0 and `RAND_MAX`, where `RAND_MAX`=2147483647. The random number is obtained by r=r/`RAND_MAX` which returns a value in [0,1]. The seed can be taken as processor' id number to guarantee the seed on each process is different. Other than random numbers in [0,1], implicit tau uses Poisson random variable, which is obtained by transforming the random numbers in $[0, 1]$. The code of Poisson random variables is from [6].

4 Numerical experiment and the study of parallelism

4.1 The example with time performance

The chemical reaction example we use here is a reversible monomolecular system,

$$\begin{aligned} S_1 + S_2 &\xrightarrow{c_1} S_4 \\ S_4 &\xrightarrow{c_2} S_1 + S_2 \\ S_1 + S_3 &\xrightarrow{c_3} S_5 \\ S_5 &\xrightarrow{c_4} S_1 + S_3 \\ S_3 + S_4 &\xrightarrow{c_5} S_6 \\ S_6 &\xrightarrow{c_6} S_3 + S_4 \\ S_2 + S_5 &\xrightarrow{c_7} S_6 \\ S_6 &\xrightarrow{c_8} S_2 + S_5 \end{aligned} \tag{4.1}$$

Here $M = 8$, $N = 6$. The initial values for $X_0 = (70, 40, 30, 50, 20, 10)'$, and $c_i = 1$ for $i = 1, \ldots, 8$. Let the final time $T = 1$. For the implicit tau method, use $tau = 0.1$.

The running time of parallel computing for a fixed sized problem can be potentially reduced by spreading the work across a group of parallel processes. The ideal behavior of the code is using p parallel processes, it would be p times fast. However, there is no communication between different processes. Therefore, the code should definitely be close the idea behavior. Hence, there is no need to study the performance of the code. In order to illustrate the advantage of the parallel code, I ran the code with the number of samples $R = 2 \times 10^6$ on different number of process for both SSA and implicit tau.

Table 1 shows the total wall clock time for SSA and implicit tau with number of samples $R = 2 \times 10^6$ with different number of processes. It quite follows the fact that the wall clock

	μ	σ	μ_ci	σ_ci
serial code	31.3691	2.9952	[31.3599,31.3785]	[2.9887,3.0018]
8 processes	31.3770	2.9913	[31.3711,31.3829]	[2.9871,2.9954]

Table 2: Mean, standard deviation and confidence intervals for SSA using serial code and 8 processes.

time reduces to half as doubling the number of processes. It also demonstrates that parallel computing is an efficient method in the application of scientific computing for such problem.

4.2 Some new ideas on the random number generator

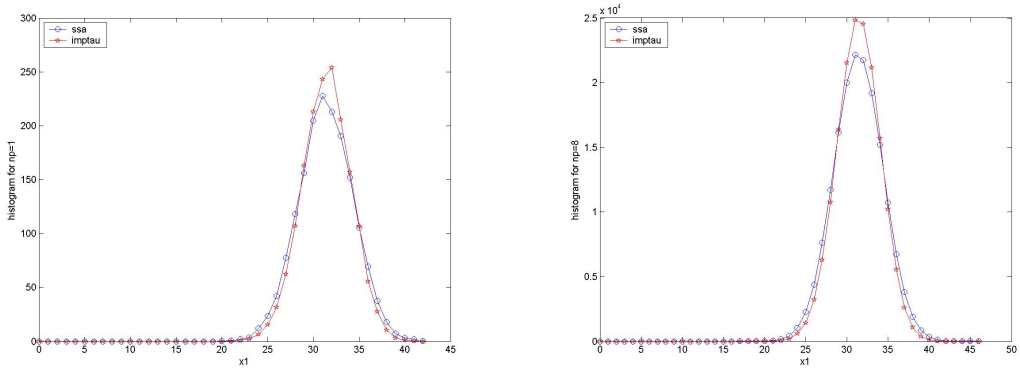
Now I will discuss the random number generator and bring about some new opinions that how to make the sequences more "random". As we know, the period of the random number generator is about 2.1×10^9. For the above example, SSA uses around 400 time steps for each simulation, and for each time step it generates 2 random numbers that gives 800 random numbers for each simulation. Since SSA is a stochastic process, by conservative estimation, it generates 10^3 random numbers per simulation. If 10^6 simulation is only distributed to one processor, then for each call, it needs about 10^9 random numbers, which is at the same degree of the cycle length.

In order to solve the issue, I use the system clock time to obtain the seed for each simulation. Here I use `gettimeofday()` function which obtains the current time, expressed as the resolution as microseconds since the Epoch. Therefore, it is guaranteed that for each simulation the seed is different, and the random numbers it requires for each call is much smaller (for the above example, 10^3) comparing the cycle. In parallel computing, there might enough synchronization between processes to get same system time, I also need the id number to calculate the system seed. The id number plus the current time in microseconds will ensure that the seeds for each process and simulation are different.

An alternative way is to use other random number generator with large cycle length. One available is Mersenne twister. It is designed to have cycle length $2^{19937} - 1$ [7].

4.3 Some statistical analysis of the data

In order to further analyze the samples, one can calculate the mean and standard deviation of each $X^{(ssa)}(T)$ and $X^{(it)}(T)$, then use some statistical tests to find the behaviors of the samples. Table 2 discusses the mean, standard deviation, 95% confidence interval for mean and standard deviation of $X^{(ssa)}(T)$ for SSA. It compares the results based on the serial and parallel code with np=8. We only study S_1, and other species should have the same behavior. Here $T = 1$. For the confidence interval of the mean and standard deviation, it shows that for serial code, the length of the confidence interval for mean is 0.01861 comparing with 0.01173 for running 8 processes. Similarly, the length of the confidence interval for standard deviation is 0.01316 comparing with 0.008291 for running 8 processes. It indicates that distributing the samples to more processes gives tighter confidence intervals, which indicates



(a) Histogram(10,000 samples) (b) Histogram(1,000,000 samples)

Figure 1: Histogram of X_1 with 10,000 samples (serial code) and 1,000,000 samples (np=8).

that we trust the mean and standard deviation more. This is a good result, and the reason is that running code on more processes takes more seeds and hence the cycle length is long enough before the random numbers start to repeat.

One way to determine whether the implicit tau method is good is to compare the histogram with SSA. Here the histogram is a a graphical display of the probability $X(T) = i$, given the number of simulations R. The probability $X(T) = i$ is denoted by $P(X = i)$. Since the initial number of each species is $X(0) = (x_1, x_2, \ldots, x_N)'$. Assume $P(X^{(ssa)} = i)$ for SSA and $P(X^{(it)} = i)$ for the implicit tau. Here $i = 0, 1, \ldots, x_T$, where $x_T = \sum_{i=1}^{N} x_i$. After generating number of samples R, compute $P(X^{(ssa)})$ and $P(X^{(it)})$ by counting the sum of the number for each $X = i$ from R samples.

Figure 1 are the comparison of the histograms from 10,000 and 1,000,000 samples for the SSA and implicit tau method. It was known that implicit tau is a good approximation to SSA, and it overestimates the variance [5]. Compare the two plots, it shows that for larger number samples, the histogram is capturing the characteristic of the true behavior more precisely, and it is smooth around the top. The histogram exhibits a good catch for large samples, which can be accomplished by using parallel code.

5 Conclusion

In this section, the experimental results show the advantages of parallel code to solve the stochastic chemical systems from the following aspects. First, it is more efficient with good time performance to solve large number of sample size. Second, even for the same size of samples, parallel code gives more reliable mean and standard deviation. Third, it captures the true stochastic behaviors better by using more samples distributing into those parallel processes.

6 Acknowledgments

The authors would like to thank Matthias Gobbert for his assistance about the parallelism of the random number generators. The hardware used in the computational studies is part of the UMBC High Performance Computing Facility (HPCF). The facility is supported by the U.S. National Science Foundation through the MRI program (grant no. CNS–0821258) and the SCREMS program (grant no. DMS–0821311), with additional substantial support from the University of Maryland, Baltimore County (UMBC). See `www.umbc.edu/hpcf` for more information on HPCF and the projects using its resources.

References

[1] Van Kampen, N.G. (2001). *Stochastic Processes in Physics and Chemistry*. North Holland.

[2] Gillespie, D.T. (1976). A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, 22, 403-434.

[3] Gillespie, D.T. (2002). The Chemical Lagngevin and Fokker-Planck equations for the reversible isomerization reaction. *Journal of Chemical Physics*, 106, 5063-5071.

[4] Gillespie, D.T. (2001). Approximation accelerated stochastic simulation of chemically reacting systems. *Journal of Chemical Physics*, 119 (24), 12784-12794.

[5] Rathinam, M., Petzold, L.R., Cao, Y., Gillespie, D.T. (2004). Stiffness in stochastic chemically reacting systems: The implicit tau-leaping method. *Journal of Chemical Physics*, 119 (24), 12784.

[6] Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P. (1992). *Numerical Recipes in C*. Cambridge University Press, second edition.

[7] Matsumoto, M., Nishimura, T. (1998). Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 3 (30).