14th Triennial World Congress, Beijing, P.R. China

DERIVATIVE ESTIMATES PARALLEL SIMULATION ALGORITHM BASED ON PERFORMANCE POTENTIALS THEORY

Chang-Chun Zou, Hong-Sheng Xi, Bao-Qun Yin, Ya-Ping Zhou, , De-Min Sun

Department of Automation, University of Science and Technology of China
Hefei, Anhui, 230027, P. R. China. Email: xihs@ustc.edu.cn

Abstract: An efficient derivative estimates parallel simulation algorithm is presented based on the new Performance Potentials theory (Cao and Chen, 1997). Two main ideas are introduced: First, a new processor-partitioning pattern, Screwy Partitioning, which can make complete load balance on a time-costing simulation part; Second, modified Common Random Number, which can remove the large amount of broadcasting cost of sample path data at the price of only adding a very little workload. The simulation experiments on an SPMD parallel computer show that this algorithm can achieve near linear speedup. *Copyright © 1999 IFAC*

Keywords: Parallel algorithms, Discrete-event dynamic systems, Performance analysis, Closed queuing networks.

1. INTRODUCTION

The stochastic theory of discrete event dynamic system (DEDS) can successfully describe the structure and performance of systems such as communication networks, manufacturing systems, traffic networks, military C^3I systems, etc. Most successful results established in DEDS performance analysis are currently dependent on infinitesimal perturbation analysis (IPA) technique, which was first proposed by Ho In early 80's and was developed by others later (Ho *et al.* 1983, Ho and Cao 1991, Cassandras 1993). Although the sample derivatives that IPA yields are unbiased or strongly consistent for many systems, it is not true for many others (Cao 1987; Heidelberger *et al.*, 1988).

Recently, Cao and Chen (1997) developed a simple approach in this direction, Markov Performance Potential theory. It provides new derivative formulas of the steady-state performance measure of Markov processes with respect to their infinitesimal generators. It is shown that the quantities involved in the derivative formulas can be easily estimated by analyzing a single sample path of a Markov process. The derivatives obtained by using these formulas can be easily proved to be unbiased and converge to their true values with probability one. Since Markov process is the most fundamental model for many discrete event systems, this potentials theory is widely applicable for sensitivity analysis of DEDS.

With respect to the fact that DEDS simulation always needs a great amount of computational effort and spends a lot of time, it is particularly important and promising to use parallel computer to speed up DEDS simulation (Righter and Walrand, 1989; Fujimoto, 1990). In this paper, for the new DEDS analysis theory: potentials theory, through analyzing the closed queuing networks simulation algorithm on single-processor, a parallel simulation algorithm is presented that promises a high parallel-efficiency.

The rest of the paper is organized as following. Section 2 briefly describes the closed queuing networks problem in Yin *et al.* (1997). In Section 3, the main formulas used in simulation are first presented, then is the simple depiction of single-

processor simulation algorithm, leaving proofs and details of performance potential theory of Cao and Chen (1997) to the original paper. In Section 4, through analyzing the single-processor simulation algorithm and serial experimental results, an insight of the main part of it is derived. Then two ideas are derived with respect to the features of the serial simulation algorithm and the characteristics of its communication. By applying these two policies, the overall parallel algorithm can be very efficient. The parallel experimental results on an SPMD parallel computer, which show that this algorithm can achieve near linear speedup, is shown in section 5. Finally, a conclusion and some discussions are given in Section 6.

2. BRIEF INTRODUCTION OF CLOSED QUEUING NETWORKS

Consider a close queuing network consisting of M servers and N customers. It is supposed that each server is equipped with a buffer that has infinite capacity and the service discipline is FCFS. The service requirement of a customer at each server is assumed to be exponentially distributed with a mean value of 1. Once a server i completes its service of a customer, this customer will immediately move from server i to any one of these M servers, including server i itself. The destination sever which the customer will move to is determined by a transition probability matrix Q — This is the simple depiction of closed queuing networks (Yin *et al.*, 1997) that are going to be discussed in this paper.

3. SERIAL SIMULATION ALGORITHM OF DERIVATIVE ESTIMATES

From the performance potentials theory, the derivatives estimate's values can be got by analyzing a single sample path of closed queuing network. Because of the limits of pages, for the proofs and details of performance potentials theory, please refer to the original paper (Cao and Chen, 1997). All the formulas and symbols in this paper are consistent with Cao and Chen (1997) and Yin *et al.* (1997).

Let J donate the performance measure. π is the steady-state probability vector of the state-transition process, while K is the state space number. g is the performance potential. f donates the performance function vector. $\mu_{i,j}$ is the service rate of the sever i at state j. A is the infinitesimal generator. The superscript $\hat{\pi}$ represents the simulation estimation value of π. Then by applying the performance potentials theory, from a single state-transition sample path, the unbiased partial derivative estimated value of J with respect to $\mu_{i,j}$ is (Cao and Chen, 1997):

$$\frac{\partial \hat{J}}{\partial \mu_{i,j}} = \hat{\pi} \frac{\partial A}{\partial \mu_{i,j}} \hat{g} + \hat{\pi} \frac{\partial f}{\partial \mu_{i,j}} \tag{1}$$

So the unbiased estimated gradient of J is:

$$\nabla \hat{J} = \left\{ \frac{\partial \hat{J}}{\partial \mu_{i,j}} \right\}_{i=1,\ j=1}^{M,\ K} \tag{2}$$

The most important work in the simulation is to record the data of events "process starts from state i, first transits to state j ", where i and j are arbitrary states. In the serial simulation program, three square $K \times K$ dimensional matrices called *parameter matrices* are used to record these kinds of data. The accordingly matrix entry (m, n) records the overall number, time summation, cost summation of the events "process starts from state i, first transits to state j " on simulation, respectively. These parameter matrices will be used to calculate the estimated value of performance potential $\hat{g}$.

The serial simulation program based on performance potentials is given below: (Use N_s donates the length of sample path in simulation, i.e. the state-transition number)

Algorithm 1. Serial simulation program based on Markov performance potentials

Step 1. Use the pseudo-random number to simulate one step of state transition.

Step 2. Update the three $K \times K$ dimensional parameter matrices. Record every state residence time and overall simulation time.

Step 3. Return to step 1 to begin next state transition until the state transition number reaches N_s.

Step 4. Use the three parameter matrices to calculate $\hat{g}$. Use every state residence time and overall simulation time to calculate $\hat{\pi}$.

Step 5. Calculate every $\frac{\partial \hat{J}}{\partial \mu_{i,j}}$ to compose the gradient matrix $\nabla \hat{J}$ (i=1,2…,M; j=1,2,…,K)

4. PARALLEL ANALYSIS AND IMPLEMENTATION

4.1. Serial simulation program analysis

It is well known that in DEDS simulation, the simulation sample path is often required to be very long. The number of state transition N_s often reaches hundreds of thousands, even millions, which is one of the reasons why parallel simulation plays an important role in DEDS research.

In order to determine which part of simulation is the pivotal part in parallel implement, many numerical experiments of serial simulation program are made on PC. The typical results are listed in table 1 and table 2.

Table 1 Fixed problem scale as $M=3, N=5, K=21$

N_s	State Trans	Matrices Compute	$\hat{g},\hat{\pi}$	$\nabla\hat{J}$
10,000	2.6%	37.3%	1.5%	58.6%
100,000	5.0%	82.0%	0.33%	12.6%
1,000,000	5.7%	92.75	0.04%	1.5%

Table 2 Fixed sample path length as N_s=100,000

M	N	K	State Trans	Matrices Compute	$\hat{g},\hat{\pi}$	$\nabla\hat{J}$
2	3	4	29.2%	69.5%	0.71%	0.60%
3	5	21	5.0%	82.0%	0.33%	12.6%
3	8	45	1.2%	72.8%	0.30%	25.7%

Remark: each column of table 1 and table 2 is computational time percentage ratio to overall simulation time.

From table 1 and table 2, it is easy to know that for middle and large scale simulation problems, there are more than 90% of overall CPU time concentrating on two parts of matrix computation:

- Computation of parameter matrices in step 2.
- Computation of gradient matrix $\nabla\hat{J}$ in step 5.

It can be seen that for large-scale simulation, parameter-matrix computation will occupy most of CPU time, even more than 90%. Therefore, the parallel implementation of step 2 and step 5, especially of the three-parameter matrices in step 2, will directly determine the overall parallel simulation program's efficiency. Therefore, these two parts are the focuses of parallel implementation.

By observing these two parts, It can be found that they have some good parallelizable feature:

- In parameter matrix computation of step 2, there are only matrix-numeral product and two matrices multiplying corresponding entries. (i.e., i column j row element in a matrix multiplies the i column j row element of another matrix)
- In step 5, each entry in matrix $\nabla\hat{J}$ is computed independently.

If the parameter matrices are partitioned with the same partitioning pattern (i.e., the i column j row's elements of these matrices are partitioned into one processor) and the gradient matrix $\nabla\hat{J}$ are partitioned evenly with arbitrary partitioning pattern, there will be no need to exchange matrices data between different processor in matrix computation.

This is a very important feature in parallel implementation, because in general parallel program, data communication cost occupies considerable run time. In order to reduce the communication cost, different parallel programs must be made for different structural parallel machines. Therefore, it can be concluded that because of the special feature of the simulation algorithm of this paper, the parallel simulation program here can achieve high parallel efficiency and can be widely applied to different structural parallel machines.

Nowadays the Multiple-Instructions Multiple-Data (MIMD) parallel machines are gaining increasingly important role in parallel area and it seems that the near-term future of parallel computing will be dominated by MIMD machines. The major disadvantages of MIMD machines such as workstation clusters are high communication latency and limited communication bandwidth. So the very low communication cost of the simulation program here makes it fitted MIMD computing well and has practical application value.

4.2. Screwy partitioning algorithm

Because the state space number K is generally large in simulation, for practicality of this parallel simulation program, the processor-partitioning pattern on matrix entries must be considered (Chen, 1994). Since there is no matrix data exchanging between different processors, only the load balance problem needs to be considered. For the gradient matrix in step 5, because each entry of it is computed independently, just dividing these $M \times K$ entries of it equally among processors will achieve a complete load balance. The essential work is for parameter matrices in step 2. From the serial simulation program, it can be seen that there are

three different types of matrix computing in step 2: some matrix's entries are computed one by one; some are computed by row and others are computed by column. General speaking, there are three different types of partitioning patterns in matrix parallel computing: block partitioning, row partitioning and column partitioning (Chen, 1994). However, for the parallel simulation algorithm here, neither of them can get a complete load balance.

Here a new partitioning method: *screwy partitioning*, is introduced. By using it, a complete load balance of these parameter matrices in step 2 can be achieved. Principle of this method is illustrated in fig.1.

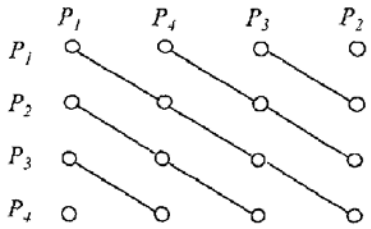

Fig.1. Illustration of screwy partitioning. Use a 4-step square matrix as example. P_i represents the ith processor. The little circles represent entries of matrix. Those circles on one line indicate that they are partitioned in one processor.

Thus, the screwy partitioning formula of matrix $[a_{i,j}](i,j=1,2,\cdots,K)$ is

$$(a_{1,1},a_{i+1,2},\cdots,a_{K,K-i+1},a_{1,K-i+2},\cdots,a_{i-1,K})\rightarrow P_i \quad (3)$$

The above formula requires that the square matrix's dimension K is equal to the number of processors. Since the state number K is often very large, in general, if using K/l processors to do simulation, the matrix entries that will partitioned into processors $P_{kl+1},P_{kl+2},\cdots,P_{kl+l}$, $(k=0,1,\cdots,K/l-1)$ according to (3) should be partitioned into processor P_{k+1}.

It should be noted that the screwy partitioning method presented here could be a general partitioning algorithm. General speaking, if the matrix computation in parallel program has both row-calculation and column-calculation, maybe by using the screwy partitioning method, a better load balance can be achieved than those ordinary-partitioning methods.

4.3. Common Random Number (CRN)

By using the screwy partitioning method, complete load balance have been achieved on two matrix computation parts that occupy more than 90% CPU time of serial simulation program. Since there will be no matrix's entries to exchange between processors in these parts, it seems that an efficient parallel algorithm has been derived. However, things are not so simple.

When developing the parallel program, it is common to let one processor to generate the single sample path, then this processor broadcasts the path's data to all other processors, who (may include the former one or not) will do parallel matrix computation together. But here in DEDS simulation, since the simulated sample path is often very long, even though there are only three number to broadcast in every state transition (old state, new state, old state residence time), the overall broadcast communication cost of sample path will be very large. This large communication cost will dramatically deteriorate the parallel efficiency of the parallel program. So an alternative method must be found to get rid of it.

In "Ordinal Optimization" (Ho, 1994), the Common Random Number (CRN) (Heidelberger and Iglehart, 1979) theme was presented to generate *N's* different sample paths for different parameters with the same random sequence number (Dai, 1996). Enlightened by it, here the CRN series can be used in all processors and let all of them to do the calculation of sample path. In this way, since the random number are identical, every processor will get the same sample path and need not to translate the path data to others. Since the sample path's computation time is less than 5% of all CPU time for most problems, the large amount of sample path's communication cost can be removed at the price of adding only a little workload. The experimental results in the following (section 5) show that this kind of modified CRN theme is the most important contribution to the high efficiency of overall parallel program.

4.4. Parallel simulation program

Now efficient parallel implementation of the main part of serial simulation program has been derived. In order to minimize the communication cost, which will dramatically deteriorate parallel performance, and simplify the parallel program, other parts of serial simulation program are not changed and just used in parallel program. The overall parallel simulation program is listed below:

Algorithm 2. Parallel simulation program based on Markov performance potentials

Step 1. In all processors, use a same random "seed" to initialize the pseudo-random number generator in each processor. In this way, the random number series in all processors will be identical to each other (CRN).

Step 2. In all processors, use the pseudo-random number to simulate one step of state transition. Since the CRN theme is used, each processor will get the same state transition result, i.e. the same sample path.

Step 3. All processors parallel compute corresponding parameter matrices' entries that are partitioned in them (use screwy partitioning method).

Step 4. Return to step 2 to begin next state transition until the state transition number reaches N_s.

Step 5. Calculate $\hat{g}, \hat{\pi}$.

Step 6. All processors parallel compute the corresponding entries of matrix $\nabla\hat{J}$. (Partition entries of matrix $\nabla\hat{J}$ equally among all processors by using any partitioning method.)

5. EXPERIMENTAL RESULTS

The computation experiments are performed on an SPMD parallel computer in Univ. Sci. & Tech. China, which is a loosely coupled message passing parallel processing computer.

To compare the performance between CRN theme and the common one without the CRN, a small-scale problem is investigated by parallel simulation: M=3, N=3, K=10, the sample path length is N_s=100,000. The results are listed in table 3 below:

Table 3 Common Random Number (CRN)

Nodes number	Use CRN theme(sec)	Not use CRN theme(sec)
1	17.0	
2	11.8	46.5
5	7.6	41.6
10	6.2	40.2

Obviously, if Common Random Number is not used in parallel program, the large amount of sample path's broadcasting cost will deteriorate performance drastically, even making the parallel program much more time consuming than serial program. It can be seen from it that the CRN theme is the most important contribution factor to the parallel program.

In order to see how well the screwy partitioning method works, a medium-scale problem (M=3, N=4, K=45, N_s=100,000) is tested based on four different partitioning methods: screwy partitioning, block, row and column partitioning. The results are listed in table 4 (time unit: second).

Table 4 Screwy partitioning method

Nodes number	Screwy	Block	Row	Column
1	261.1			
5	54.9	56.2	55.6	56.8
15	20.1	20.4	21.5	22.0

It can be seen from table 4 that in practice the screwy partitioning method does not play an important role in parallel program. This is reasonable after analyzing the parallel program. The screwy partitioning method is used for parallel processing the parameter metrics computation in step 2 of serial program. In step 2, the row and column computation is addition and subtraction, while the one by one computation is multiplication. So the row and column computation workload is much smaller than that of matrix's entries one by one computation, for which part these different partitioning methods make no differences.

Since in most cases, the parallel simulation program is only used for large-scale problems, it is necessary to do experiments for large-scale problems to see the parallel program's performance. For a large-scale case: M=5, N=8, K=495, together with the above two cases, the experiment results are summarized in table 5 (Use screwy partitioning method. The sample path lengths in all cases are N_s=100,000).

Table 5 The experimental results of parallel simulation algorithm

Cases scale	Nodes number	Time (sec)	Speedup	Efficiency
K=10	1	17.0		
	2	11.8	1.44	72.0%
	5	7.6	2.24	44.8%
	10	6.2	2.74	27.4%
K=45	1	261.1		
	5	54.9	4.76	95.2%
	15	20.8	12.55	83.7%
K=	1	32546		
495	3	11172	2.91	97.1%
	5	6687	4.87	97.3%
	9	3755	8.67	96.3%

Obviously, the parallel simulation program here exhibits excellent speedup and scalability. Furthermore, the larger the case's scale is, the better performance it behaves, this makes it suitable and effective to practical usage.

CONCLUSION

Based on a new DEDS analysis theory: Markov

performance potentials (Cao and Chen, 1997), an efficient parallel simulation algorithm is provided when applying the potentials theory on a class of closed queuing networks. The experimental results show that the parallel simulation algorithm of this paper exhibits excellent speedups and scalability.

In this paper, two ideas are presented, Screwy Partitioning method and modified Common Random Number theme. Although the screwy partitioning method does not play a very important role in the parallel program, it is a kind of general partitioning method that can be used in many other parallel implementation areas.

There are still many topics for future research. In this parallel simulation program, based on the characteristics of the potential theory and closed queuing network, the matrix computation parts of the counterpart serial program are parallel implemented and get a high efficiency. But for general DEDS parallel simulation, researchers are mainly doing the 'sample path' parallel simulation (Righter and Walrand, 1989; Fujimoto, 1990) and now studying simulation of many different DEDS simultaneously on parallel computers. (Vakili 1991,1992; Hu, 1995) Since the recently developed concept of ordinal optimization (Ho, 1996) has become popular and promising, it is likely a prospective research direction by combining the derivative estimates parallel algorithm here with the many-sample-path DEDS parallel simulation and the ordinal optimization.

ACKNOWLEDGEMENTS

The authors thank China High Performance Computing Center at Hefei to provide the SPMD parallel computer for the experiments.

REFERENCES

Cao, X.R. (1987). First-order perturbation analysis of a single multi-class finite source queue, *Performance Evaluation*, **7**, 31-41.

Cao, X.R. (1994). *Realization Probabilities: the Dynamics of Queuing Systems*, Springer-Verlag, New York.

Cao, X.R. and H.F. Chen (1997). Perturbation realization, potentials and sensitivity analysis of Markov processes, *IEEE Transactions on Automatic Control*, **42**, 1382-1393.

Cassandras, C.G. (1993). *Discrete Event Systems: Modeling and Performance Analysis*, Aksen Associates, Inc.

Chen, G.L. (1994). *Design and Analysis of parallel algorithm*, High Educational Publishing House, PR China.

Dai, L.Y. (1996). Convergence Properties of Ordinal Comparison in the Simulation of Discrete Event Dynamic Systems, *Journal of Optimization Theory and Applications*, **91**, 363-388

Fujimoto, R.M. (1990). Parallel discrete event simulation, *Commu. ACM*, **33**, 31-53.

Heidelberger, P. and D.L. Iglehart (1979). Comparing stochastic systems using regenerative simulation with common random numbers, *Adv. Appl. Prob.*, **11**, 804-819.

Heidelberger, P., X.R. Cao, M. Zazanis and R. Suri (1988). Convergence properties of infinitesimal perturbation analysis estimates, *Management Science*. **34**, 1281-1302.

Ho, Y.C., X.R. Cao and C.G.Cassandras (1983). Infinitesimal and Finite Perturbation Analysis for Queueing Networks, *Automatica*, **19**, 439-445.

Ho, Y.C. and X.R. Cao (1991). In: *Perturbation analysis of discrete event dynamic systems*, Kluwer Academic Publisher, Boston.

Ho, Y.C. (1994). Overview of Ordinal Optimization, *Proceedings of the 33rd Conference on Decision and Control*. Lake Buena Vista, Florida, 1975-1977.

Ho, Y.C. (1996). Soft Optimization for Hard Problems, In: *Computerized Lecture via private communication.*

Hu, J.Q. (1995). Parallel Simulation of DEDS Via Event Synchronization. *DEDS: Theory and Applications*, **5**, 167-186.

Righter, R. and J.C.Walrand (1989). Distributed simulation of discrete event systems, *Proceedings of the IEEE* **77**, 99-113.

Ross, S.M. (1983). *Stochastic Processes*. Wiley.

Vakili, P. (1991). Using a standard clock technique for efficient simulation. *Operations Research Letters*, **10**, 445-452.

Vakili, P. (1992). Massively parallel and distributed simulation of a class of discrete event systems: A different perspective. *ACM Trans. On Modeling and Simulation*, **2**, 214-238.

Whitt, W. (1980). Continuity of generalized semi-Markov processes. *Mathematics of Operations Research*, **5**, 494-501.

Yin, B.Q., Y.P. Zhou, H.S. Xi and D.M. Sun (1997). Sensitivity formulas of performance and realization factors with parameter-dependent performance functions in closed queuing networks, In: *Proceedings of CWCICIA*, 1884-1889. Xi'an Jiaotong Univ. Press, PR China.