Coarse Grained Parallel Monte Carlo Algorithms for Solving SLAE Using PVM

V. Alexandrov[1], F. Dehne[2], A. Rau-Chaplin[3], and K. Taft[1]

[1] Department of Computer Science, University of Liverpool,
Chadwick Building, Peach Street, Liverpool, L69 7ZF, UK
{vassil, keitht} @csc.liv.ac.uk
[2] School of Computer Science, Carleton University,
Ottawa, Canada, K1S 5B6
dehne@scs.carleton.ca
[3] Faculty of Computer Science, DalTech, Dalhousie University,
P.O. Box 1000, Halifax NS, Canada B3J 2X4
arc@cs.dal.ca

Abstract. The problem of solving System of Linear Algebraic Equations (SLAE) by parallel Monte Carlo numerical methods is considered. Three Monte Carlo algorithms are presented. In case when copy of the matrix is sent to each processor the execution time for solving SLAE by Monte Carlo on p processors is bounded by $O(nNT/p)$ (excluding the initial loading of the data) where N is the number of chains and T is the length of the chain in the stochastic process, which are independent of matrix size n.
Numerical tests are performed for a number of dense and sparse test matrices using PVM on a cluster of workstations.

1 Introduction

It is known that Monte Carlo methods give statistical estimates for the components of the solution vector of SLAE by performing random sampling of a certain random variable whose mathematical expectation is the desired solution [9,10]. We consider Monte Carlo methods for solving SLAE since: **firstly,** only $O(NT)$ steps are required to find an element of the inverse matrix (MI) or component of the solution vector of SLAE (N is a number of chains and T is a measure on the chains length in the stochastic process, which are independent of n) and **secondly,** the sampling process for stochastic methods is inherently parallel. In comparison, the direct methods of solution require $O(n^3)$ sequential steps when the usual elimination or annihilation schemes (e.g non-pivoting Gaussian Elimination, Gauss-Jordan methods) are employed [3]. Consequently the computation time for very large problems or for real-time problems can be prohibitive and prevents the use of many established algorithms. Therefore due to their properties, their inherent parallelism and loose data dependencies Monte Carlo algorithms can be implemented on parallel machines very efficiently and thus may enable us to solve large-scale problems which are sometimes difficult or prohibitive to be solved by the well-known numerical methods.

V. Alexandrov and J. Dongarra (Eds.): PVM/MPI'98, LNCS 1497, pp. 323–330, 1998.

Generally three Monte Carlo methods for Matrix Inversion (MI) and finding a solution vector of System of Linear Algebraic Equations (SLAE) can be outlined: with absorption, without absorption with uniform transition frequency function, and without absorption with almost optimal transition frequency function.

In the case of **fine grained** setting, recently Alexandrov, Megson and Dimov have shown that an $n \times n$ matrix can be inverted in $3n/2+N+T$ steps on regular array with $O(n^2NT)$ cells [8]. Alexandrov and Megson have also shown that a solution vector of SLAE can be found in $n + N + T$ steps on regular array with the same number of cells [2]. A number of bounds on N and T have been established, which show that these designs are faster than the existing designs for large values of n [8,2].

The **coarse grained** case for MI is considered in [1]. In this paper we extend this implementation approach for SLAE in MIMD environment, i.e. a cluster of workstations under PVM in our case. We also derive an estimate on time complexity using CGM model.

The **Coarse Grained Multicomputer** model, or $CGM(n, p)$ for short, which is the architectural model to be used in this paper is a set of p processors with $O(\frac{n}{p})$ local memory each, connected to some arbitrary interconnection network or a shared memory. The term "**coarse grained**" refers to the fact that (as in practice) the size $O(\frac{n}{p})$ of each local memory is defined to be "considerably larger" than $O(1)$. Our definition of "considerably larger" will be that $\frac{n}{p} \geq p$. This is clearly true for all currently available coarse grained parallel machines. For determining time complexities we will consider both, local computation time and inter-processor communication time, in the standard way.

For parallel algorithms for SLAE to be relevant in practice, such algorithms must be **scalable**, that is, they must be applicable and efficient for a wide range of ratios $\frac{n}{p}$. The use of CGM helps to ensure that the parallel algorithms designed are not only efficient in theory, but also they result in efficient parallel software with fast running time on real data . Experiments have shown that in addition to the scalability, the CGM algorithms typically quickly reach the point of optimal speedup for reasonable data sets. Even with modest programming efforts the actual results obtained for other application areas have been excellent [4].

In this paper we focus mainly on the case when a copy of the matrix is sent to each processor. We are currently testing different strategies for efficiently parallelising Monte Carlo algorithms for the case of large matrices when the matrix is partitioned and distributed among the processors as well as different ways of minimising the number of chains required to find the solution. We expect to report these results in the near future.

2 Stochastic Methods and SLAE

Assume that the system of linear algebraic equations (SLAE) is presented in the form:

$$x = Ax + \varphi \tag{1}$$

where A is a real square $n \times n$ matrix, $x = (x_1, x_2, ..., x_n)^t$ is a $1 \times n$ solution vector and $\varphi = (\varphi_1, \varphi_2, ..., \varphi_n)^t$ is a given vector. (If we consider the system $Lx = b$, then it is possible to choose non-singular matrix M such that $ML = I - A$ and $Mb = \varphi$, and so $Lx = b$ can be presented as $x = Ax + \varphi$.) Assume that A satisfies the condition $\max_{1 \le i \le n} \sum_{j=1}^{n} |a_{ij}| < 1$, which implies that all the eigenvalues of A lie in the unit circle. The matrix and vector norms are determined as follows: $\|A\| = \max_{1 \le i \le n} \sum_{j=1}^{n} |a_{ij}|$, $\|\varphi\| = \max_{1 \le i \le n} |\varphi_i|$.

Suppose that we have Markov chains with n - states. The random trajectory (chain) T_i of length i starting in state k_0 is defined as $k_0 \to k_1 \to \cdots \to k_j \to \cdots \to k_i$ where k_j is the number of the state chosen, for $j = 1, 2, \cdots, i$. The following probability definitions are also important: $P(k_0 = \alpha) = p_\alpha, P(k_j = \beta | k_{j-1} = \alpha) = p_{\alpha\beta}$ where p_α is the probability that the chain starts in state α and $p_{\alpha\beta}$ is the transition probability to state β from state α. Probabilities $p_{\alpha\beta}$ define a transition matrix P. We require that $\sum_{\alpha=1}^{n} p_\alpha = 1$, $\sum_{\beta=1}^{n} p_{\alpha\beta} = 1$ for any $\alpha = 1, 2, ..., n$, the distribution $(p_1, ..., p_n)^t$ is acceptable to vector g and similarly the distribution $p_{\alpha\beta}$ is acceptable to A [9].

Consider the problem of evaluating the inner product of a given vector g with the vector solution of (1)

$$(g, x) = \sum_{\alpha=1}^{n} g_\alpha x_\alpha \tag{2}$$

It is known [9] that the mathematical expectation $E\Theta^*[g]$ of random variable $\Theta^*[g]$ is:

$$\begin{aligned} E\Theta^*[g] &= (g, x) \\ \text{where } \Theta^*[g] &= \frac{g_{k_0}}{p_{k_0}} \sum_{j=0}^{\infty} W_j \varphi_{k_j} \\ \text{and} \quad W_0 &= 1 \,, \quad W_j = W_{j-1} \frac{a_{k_{j-1} k_j}}{p_{k_{j-1} k_j}} \end{aligned} \tag{3}$$

We use the following notation for a partial sum (3) $\theta_i[g] = \frac{g_{k_0}}{p_{k_0}} \sum_{j=0}^{i} W_j \varphi_{k_j}$. According to the above conditions on the matrix A, the series $\sum_{j=0}^{\infty} W_j \varphi_{k_j}$ converges for any given vector φ and $E\theta_i[g]$ tends to (g, x) as $i \longrightarrow \infty$. Thus $\theta_i[g]$ can be considered an estimate of (g, x) for i sufficiently large.

Now we define the Monte Carlo method. To find one component of the solution, for example the r-th component of x, we choose $g = e(r) = (0, ..., 0, 1, 0, ..., 0)$ where the one is in the r-th place. It follows that $(g, x) = \sum_{\alpha=1}^{n} e_\alpha(r) x_\alpha = x_r$ and the corresponding Monte Carlo method is given by

$$x_r \approx \frac{1}{N} \sum_{s=1}^{N} \theta_i[e(r)]_s \tag{4}$$

where N is the number of chains and $\theta_i[e(r)]_s$ is the value of $\theta_i[e(r)]$ in the s-th chain.

The **probable error** of the method, is defined as $r_N = 0.6745\sqrt{D\theta/N}$, where $P\{|\bar{\theta} - E(\theta)| < r_N\} \approx 1/2 \approx P\{|\bar{\theta} - E(\theta)| > r_N\}$, if we have N independent realizations of random variable (r.v.) θ with mathematical expectation $E\theta$ and average $\bar{\theta}$ [9].

It is clear from the formula for r_N that the number of chains N can be reduced by a suitable choice of the transition probabilities that reduces the variance for a given probable error. This idea leads to Monte Carlo methods with minimal probable error.

The key results concerning minimization of probable error and the definition of **almost** optimal transition frequency for Monte Carlo methods applied to the calculation of inner product via iterated functions are presented in [8]. According to [8,7] and the principal of collinearity of norms [8] we can choose $p_{\alpha\beta}$ proportional to the $|a_{\alpha\beta}|$.

In case of Monte Carlo with absorption, assuming as before $\|A\| < 1$, we have [1,2,7]:

$$p_{\alpha\beta} = |a_{\alpha\beta}| \text{ for } \alpha, \beta = 1, 2, ..., n.$$

and the absorption probability

$$p_{\alpha n+1} = p_\alpha = 1 - \sum_{\beta=1}^{n} p_{\alpha\beta}$$

is the probability that the trajectory ends in state α.

In case of Monte Carlo without absorption we have two possibilities [1,2] :

– Almost Optimal Monte Carlo method:

$$p_{\alpha\beta} = \frac{|a_{\alpha\beta}|}{\sum_\beta |a_{\alpha\beta}|} \text{ for } \alpha, \beta = 1, 2, ..., n.$$

– Usual Monte Carlo method:

$$p_{\alpha\beta} = 1/n \text{ for } \alpha, \beta = 1, 2, ..., n$$

In case of $\|A\| \geq 1$ or very close to 1 we can use the Resolvent Monte Carlo method [5] to reduce the matrix norm and to speedup the computations.

3 Parameters Estimation and Discussion

We will outline the method of estimation of N and T in case of **Monte Carlo method without absorbing states** since it is known that these methods require less chains than the methods with absorption to reach the same precision [2,1]. In case of Monte Carlo with absorption the parameter estimation can be done in the same way. We will consider Monte Carlo methods with uniform (UM) and with almost optimal (MAO) transition frequency function. Let us consider Monte Carlo method with almost optimal (MAO) transition frequency function. We assume that the following conditions $\sum_{\beta=1}^{n} p_{\alpha\beta} = 1$ for any $\alpha = 1, 2, ..., n$

must be satisfied and transition matrix P might have entries $p_{\alpha\beta} = \frac{|a_{\alpha\beta}|}{\sum_{\beta}|a_{\alpha\beta}|}$ for $\alpha, \beta = 1, 2, ..., n$.

The estimator Θ^* for SLAE was defined as follows

$$\begin{aligned} E\Theta^*[g] &= (g, x), \\ \text{where } \Theta^*[g] &= \frac{g_{k_0}}{p_{k_0}} \sum_{j=0}^{\infty} W_j \varphi_{k_j} \\ \text{and} \quad W_0 = 1\ , &\quad W_j = W_{j-1} \frac{a_{k_{j-1}k_j}}{p_{k_{j-1}k_j}}. \end{aligned} \tag{5}$$

The sum for Θ^* must be dropped when $|W_i \varphi_{k_i}| < \delta$ [9].

Note that

$$|W_i \varphi_{k_i}| = |\frac{a_{\alpha_0\alpha_1} \cdots a_{\alpha_{i-1}\alpha_i}}{\frac{|a_{\alpha_0\alpha_1}|}{\|A\|} \cdots \frac{|a_{\alpha_{i-1}\alpha_i}|}{\|A\|}}||\varphi_{k_i}| = \|A\|^i \|\varphi\| < \delta.$$

Then it follows that

$$T = i \leq \frac{\log(\delta/\|\varphi\|)}{\log\|A\|}.$$

It is easy to find [9] that $|\Theta^*| \leq \frac{\|\varphi\|}{(1-\|A\|)}$, which means that variance of r.v. Θ^* is bounded by its second moment: $D\Theta^* \leq E\Theta^{*2} = \frac{\|\varphi\|^2}{(1-\|A\|)^2} \leq \frac{f^2}{(1-\|A\|)^2}$. According to the Central Limit Theorem for the given error ϵ

$$N \geq \frac{0.6745^2 D\eta^*[g]}{\epsilon^2} \quad \text{and} \quad \text{thus } N \geq \frac{0.6745^2}{\epsilon^2} \frac{f^2}{(1-\|A\|)^2} \tag{6}$$

is a lower bound on N which is independent of n.

It is clear that T and N depend only on the matrix norm and precision. Furthermore, the size of N can be controlled by an appropriate choice of ϵ once P and A are known.

Consider N and T as functions of $\frac{1}{(1-\|A\|)}$. It is obvious from (6) that $T = O(\sqrt{(N)})$. In addition there are computational experiments in [8] showing this fact that for sufficiently large N we can take $T \approx \sqrt{N}$.

4 Parallel Implementation

We implement parallel Monte Carlo algorithms on a cluster of workstations under PVM. We assume virtual star topology and we apply master/slave approach.

Inherently, Monte Carlo methods for solving SLAE allow us to have minimal communication, i.e. to pass the full matrix A to every processor, to run the algorithm in parallel on each processor computing $\lceil n/p \rceil$ components of the solution vector and to collect the results from slaves at the end without any communication between sending A and receiving partitions of x. The only communication is at the beginning and at the end of the algorithm execution which allows us to obtain very high efficiency of parallel implementation. Therefore, by allocating

the master in the central node of the star and the slaves in the remaining nodes, the communication is minimized.

Therefore since we need to compute n components of the vector solution each requiring N chains of length T on p processors in parallel the time is $O(nNT/p)$ excluding the initial loading of the data.

5 Numerical Tests

The numerical tests are made on a cluster of 48 Hewlett Packard 900 series 700 Unix workstations under PVM. The workstations are networked via 10Mb switched ethernet segments and each workstation has at least 64Mb RAM and run at least 60 MIPS.

The numerical tests are made using methods without absorption, since they require less chains to reach given precision in comparison with methods with absorption [1]. We have used dense and sparse balanced matrices (which have nearly equal sums of elements per row). In the example presented, the matrix is dense with norm 0.98 when the convergence of the method is slow. The results for the average time and efficiency are given in tables 1 and 2. The relative accuracy is 10^{-2}.

Table 1. Time in seconds

Processors / **Matrix Size**	1	2	3	4	6	8	10
50	36.108	18.159	12.359	9.153	7.427	5.419	4.006
100	286.318	143.397	104.428	72.178	52.302	40.271	32.93
200	581.628	286.244	192.719	144.356	105.609	80.828	62.699
300	855.506	445.109	358.114	229.093	162.159	124.316	103.430
400	1150.567	574.193	385.461	302.706	219.099	174.823	166.83
500	1494.473	741.936	564.527	409.003	280.916	224.696	210.679

The experiments show that the computation time is a **linear function** of the matrix size n which is in accordance with the theoretical estimates.

The **parallel efficiency** E as, a measure that characterize the quality of the proposed algorithms is defined as:

$$E(X) = \frac{ET_1(X)}{pET_p(X)},$$

where X is a Monte Carlo algorithm, $ET_p(X)$ is the expected value of the computational time for implementation the algorithm X on a system of p processors.

Table 2. Efficiency

Processors / **Matrix Size**	1	2	3	4	6	8	10
50	1	0.994	0.974	0.986	0.810	0.833	0.9013
100	1	0.9985	0.914	0.992	0.912	0.889	0.869
200	1	1.016	1.006	1.007	0.918	0.8995	0.928
300	1	0.961	0.796	0.9335	0.879	0.86	0.827
400	1	1.0015	0.995	0.95	0.875	0.823	0.6897
500	1	1.007	0.891	0.9135	0.877	0.831	0.7094

6 Conclusion

In our parallel implementation we have to compute n components of the solution vector of SLAE in parallel. To compute a component of the solution vector we need N independent chains with length T, and for n components in parallel we need nN such independent chains of length T, where N and T are the mathematical expectations of the number of chains and chain length, respectively. So the execution time on p processors for solving SLAE by Monte Carlo is bounded by $O(nNT/p)$ (excluding initialization communication time). According to the discussion and results above N and T depend only on the matrix norm and precision and do not depend on the matrix size. Therefore the Monte Carlo methods can be efficiently implemented on MIMD environment and in particular on a cluster of workstations under PVM.

In particular it should be noted that the Monte Carlo methods are well suited to large problems where other solution methods are impractical or impossible for computational reasons, for calculating quick rough estimate of the solution vector, and when only a few components of the solution vector are desired. Consequently, if massive parallelism is available and if low precision is acceptable, Monte Carlo algorithms could become favourable for $n >> N$.

7 Acknowledgements

Thanks to the British Council for the partial support.

References

1. V.Alexandrov and S. Lakka *Comparison of three Parallel Monte Carlo Methods for Matrix Inversion*, Proc. of EUROPAR96, Lyon, France, Vol II, pp. 72-80.
2. V.Alexandrov and G.M. Megson *Solving Sytem of Linear algebraic Equations by Monte Carlo Method on Regular Arrays*, Proc. of PARCELLA96, 16-20 September, Berlin, Germany, pp. 137-146, 1996.
3. Bertsekas D.P. and Tsitsiklis , *Parallel and Distributed Computation* , Prentice Hall, 1989

4. F. Dehne, A. Fabri, and A. Rau-Chaplin, Scalable parallel geometric algorithms for multicomputers, *Proc. 7th ACM Symp. on Computational Geometry*, 1993.
5. I. Dimov and V.Alexandrov A New Highly Convergent Monte Carlo Method for Matrix Computations, Proc. of IMACS Monte Carlo Seminar, April 1-4, 1997, Belgium (in print).
6. G. H. Golub, Ch. F. Van Loon, *Matrix Computations*, **The Johns Hopkins Univ. Press**, Baltimore and London, 1996.
7. J.H. Halton, *Sequential Monte Carlo Techniques for the Solution of Linear Systems*, **TR 92-033**, University of North Carolina at Chapel Hill, Department of Computer Science, 46 pp., 1992.
8. G.M.Megson, V.Aleksandrov, I. Dimov *Systolic Matrix Inversion Using Monte Carlo Method,* J. Parallel Algorithms and Applications , Vol.3, pp.311-330, 1994.
9. Sobol' I.M. *Monte Carlo numerical methods.* Moscow, Nauka, 1973 (Russian)(English version Univ. of Chicago Press 1984).
10. Westlake J.R., *A Handbook of Numerical Matrix Inversion and Solution of Linear Equations*, John Wiley and Sons, New York, 1968.