Parallel Monte Carlo Algorithms for Sparse SLAE Using MPI

V. Alexandrov[1] and A. Karaivanova[2]

[1] Department of Computer Science, University of Liverpool
Chadwick Building, Peach Street, Liverpool, L69 7ZF, UK
vassil@csc.liv.ac.uk

[2] Central Laboratory for Parallel Processing, Bulgarian Academy of Sciences
Acad. G. Bonchev St.,bl. 25 A, 1113, Sofia, Bulgaria
anet@copern.acad.bg

Abstract. The problem of solving sparse Systems of Linear Algebraic Equations (SLAE) by parallel Monte Carlo numerical methods is considered. The almost optimal Monte Carlo algorithms are presented. In case when a copy of the non-zero matrix elements is sent to each processor the execution time for solving SLAE by Monte Carlo on p processors is bounded by $O(nNdT/p)$ where N is the number of chains, T is the length of the chain in the stochastic process, which are independent of matrix size n, and d is the average number of non-zero elements in the row. Finding a component of the solution vector requires $O(NdT/p)$ time on p processors, which is independent of the matrix size n.

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 [11,12]. 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 for dense matrices when the usual elimination or annihilation schemes (e.g non-pivoting Gaussian Elimination, Gauss-Jordan methods) are employed [4]. While considering general sparse matrices a reordering algorithms are usually used before applying direct or iterative methods of solution. The time required for reordering is $O(n^2)$ for straightforward reordering or $O(Z\,log(n))$ if binary trees are used [13], where Z is the number of non-zero elements in the matrix.

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.

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 [10]. 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 [10,2].

The **coarse grained** case for MI is considered in [1]. The **coarse grained** parallel Monte Carlo algorithms for solving dense SLAE and finding a dominant eigenvalue are considered in [3] and [6] respectively. In this paper we extend this implementation approach for sparse SLAE in MIMD environment, i.e. a cluster of workstations under MPI 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 [5].

In this paper we focus mainly on the case when a copy of the non-zero elements of a sparse matrix is sent to each processor.

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 [11].

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 [11] 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 $\bar{\theta}$ [11].

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 [10]. According to [10,9] and the principal of collinearity of norms [10] we can choose $p_{\alpha\beta}$ proportional to the $|a_{\alpha\beta}|$.

In case of Almost Optimal Monte Carlo [1,2] the following transition probabilities $p_{\alpha\beta}$ are applied :

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

In case of $\|A\| \geq 1$ or very close to 1 we can use the Resolvent Monte Carlo method [7] 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 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

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

The sum for Θ^* must be dropped when $|W_i \varphi_{k_i}| < \delta$ [11].

Note that

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

Then it follows that

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

It is easy to find [11] 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

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

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.

4 Parallel Implementation

We implement parallel Monte Carlo algorithms on a cluster of workstations under MPI. 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 non-zero elements of the sparse 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 non-zero elements of A and receiving partitions of x. Even in the case we compute only k components $(1 \leq k \leq n)$ of the solution vector we can divide evenly the number of chains among the processors, e.g. distributing $\lceil kN/p \rceil$ chains on each processor. 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.

Since we need to compute n components of the vector solution each requiring N chains of length T and having d non-zero elements in average in a row of a given sparse matrix on p processors in parallel, the time is $O(nNdT/p)$. This estimate includes logical operations, multiplications and additions.

5 Numerical Tests

The numerical tests are made on a cluster of 48 Hewlett Packard 900 series 700 Unix workstations under MPI (version 1.1). The workstations are networked via 10Mb switched ethernet segments and each workstation has at least 64Mb RAM and run at least 60 MIPS.

We have carried out several type of experiments. First, we have considered the case when one component of the solution vector is computed. In this case each processor executes the same program for N/p number of trajectories, i.e. it computes N/p independent realizations of the random variable . At the end the host processor collects the results of all realizations and computes the desired value. The computational time does not include the time for initial loading of the matrix because we consider our problem as a part of bigger problem (for example, *spectral portraits of matrices*) and suppose that every processor constructs it.

Second we have considered computing an inner product (h, x).

The **parallel efficiency** E 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 1. Implementation of the **Monte Carlo Algorithm** using MPI for calculating one component of the solution (number of trajectories - 100000).

	1*pr.*	2*pr.*	2*pr.*	4*pr.*	4*pr.*	5*pr.*	5*pr.*	8*pr.*	8*pr.*
	$T(ms)$	$T(ms)$	E	$T(ms)$	E	$T(ms)$	E	$T(ms)$	E
SLAE n = 128	38	19	1	15	0.63	12	0.63	8	0.6
SLAE n = 1024	28	14	1	8	0.9	6	0.93	4	0.9
SLAE n = 2000	23	11	1.04	6	0.96	5	0.92	5	0.6

Table 2. Implementation of the **Monte Carlo Algorithm** for evaluation of the scalar product (h, x), where x is the unknown solution vector, and h is a given vector, using MPI (number of trajectories - 100000).

	1*pr.*	2*pr.*	2*pr.*	4*pr.*	4*pr.*	5*pr.*	5*pr.*	8*pr.*	8*pr.*
	$T(ms)$	$T(ms)$	E	$T(ms)$	E	$T(ms)$	E	$T(ms)$	E
SLAE n = 128	16	8	1	4	1	3	1.01	3	0.7
SLAE n = 1024	105	53	0.98	28	0.94	21	1	14	0.94
SLAE n = 2000	167	84	0.99	58	0.7	42	0.8	33	0.7

In all cases the test matrices are sparse and are stored in **packed row format** (i.e. only non-zero elements). The average number of non-zero elements per matrix row is $d = 52$ for $n = 128$, $d = 57$ for $n = 1024$ and $d = 56$ for $n = 2000$ respectively. The results for average time and efficiency are given in tables 1 and 2 and look promising. The relative accuracy is 10^{-3}.

As you can see in case when one component of the vector solution is computed the time does not depend on the matrix size n. When all the components of the solution vector are computed the time is linear of the matrix size n. This corroborates our theoretical results.

6 Conclusion

In our parallel implementation we have to compute n components of the solution vector of sparse SLAE in parallel. To compute a component of the solution vector we need N independent chains with length T for matrix with d non-zero elements per row in average, 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 $O(nNdT/p)$ (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 MPI.

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. This method also do not require any reordering strategies to be implemented. Consequently, if massive parallelism is available and if low precision is acceptable, Monte Carlo algorithms could become favorable for $n >> N$.

References

1. Alexandrov, V., Lakka, S.: Comparison of three Parallel Monte Carlo Methods for Matrix Inversion, Proc. of EUROPAR96, Lyon, France, Vol II (1996), 72-80
2. Alexandrov, V., Megson, G.M.: Solving Sytem of Linear algebraic Equations by Monte Carlo Method on Regular Arrays, Proc. of PARCELLA96, 16-20 September, Berlin, Germany, (1996) 137-146
3. Alexandrov, V., Rau-Chaplin, A., Dehne, F., Taft, K.: Efficient Coarse Grain Monte Carlo Algorithms for Matrix Computations using PVM, LNCS 1497, Springer, August, (1998) 323-330
4. Bertsekas, D.P., Tsitsiklis : Parallel and Distributed Computation, Prentice Hall, (1989)
5. Dehne, F., Fabri, A., Rau-Chaplin, A.: Scalable parallel geometric algorithms for multicomputers, Proc. 7th ACM Symp. on Computational Geometry, (1993)

6. Dimov, I., Alexandrov, V., Karaivanova, A.: Implementation of Monte Carlo Algorithms for Eigenvalue Problem using MPI, LNCS 1497, Springer, August (1998) 346-353
7. Dimov, I., Alexandrov, V.: A New Highly Convergent Monte Carlo Method for Matrix Computations, Mathematics and Computers in Simulation, Vol. **47**, No 2-5, North-Holland, August (1998) 165-182
8. Golub, G.H., Ch., F., Van Loon: Matrix Computations, The Johns Hopkins Univ. Press, Baltimore and London, (1996)
9. Halton, J.H.: Sequential Monte Carlo Techniques for the Solution of Linear Systems, TR 92-033, University of North Carolina at Chapel Hill, Department of Computer Science, (1992)
10. Megson,G.M., Aleksandrov, V., Dimov, I.: Systolic Matrix Inversion Using Monte Carlo Method, J. Parallel Algorithms and Applications , Vol.**3**, (1994) 311-330
11. Sobol', I.M.: Monte Carlo numerical methods. Moscow, Nauka, (1973) (Russian)(English version Univ. of Chicago Press 1984).
12. Westlake J.R.: A Handbook of Numerical Matrix Inversion and Solution of Linear Equations, John Wiley and Sons, New York, (1968)
13. Gallivan,K., Hansen,P.C., Ostromsky, Tz., Zlatev, Z.: A Locally Optimized Reordering Algorithm and its Application to a Parallel Sparse Linear System Solver, Computing V.**54**, (1995) 39-67