Parallel Hybrid Monte Carlo Algorithms for Matrix Computations

V. Alexandrov[1], E. Atanassov[2], I. Dimov[2], S.Branford[1], A. Thandavan[1] and C. Weihrauch[1]

[1]Department of Computer Science, University of Reading
[2]IPP, Bulgarian Academy of Sciences

Abstract

In this paper we consider hybrid (fast stochastic approximation and deterministic refinement) algorithms for Matrix Inversion (MI) and Solving Systems of Linear Equations (SLAE). Monte Carlo methods are used for the stochastic approximation, since it is known that they are very efficient in finding a quick rough approximation of the element or a row of the inverse matrix or finding a component of the solution vector. We show how the stochastic approximation of the MI can be combined with a deterministic refinement procedure to obtain MI with the required precision and further solve the SLAE using MI. We employ a splitting $A = D - C$ of a given non-singular matrix A, where D is a diagonal dominant matrix and matrix C is a diagonal matrix. In our algorithm for solving SLAE and MI different choices of D can be considered in order to control the norm of matrix $T = D^{-1}C$, of the resulting SLAE and to minimize the number of the Markov Chains required to reach given precision. Further we run the algorithms on a mini-Grid and investigate their efficiency depending on the granularity. Corresponding experimental results are presented.

Keywords: Monte Carlo Method, Markov Chain, Matrix Inversion, Solution of System of Linear Equations, Grid Computing

1 Introduction

The problem of inverting a real $n \times n$ matrix (MI) and solving system of linear algebraic equations (SLAE) is of an unquestionable importance in many scientific and engineering applications: e.g. communication, stochastic modelling, and many physical problems involving partial differential equations. For example, the direct parallel methods of solution for systems with dense matrices require $O(n^3/p)$ steps when the usual elimination schemes (e.g. non-pivoting Gaussian elimination, Gauss-Jordan methods) are employed [4]. Consequently the computation time for very large problems or real time problems can be prohibitive and prevents the use of many established algorithms.

It is known that Monte Carlo methods give statistical estimation of the components of the inverse matrix or elements of the solution vector by performing random sampling of a certain random variable, whose mathematical expectation is the desired solution. We concentrate on Monte Carlo methods for MI and solving SLAEs, since, firstly, only $O(NL)$ steps are required to find an element of the inverse matrix where N is the number of chains and T is an estimate of the chain length in the stochastic process, which are independent of matrix size n and secondly, the process for stochastic methods is inherently parallel.

Several authors have proposed different coarse grained Monte Carlo parallel algorithms for MI and SLAE [7, 8, 9, 10, 11]. In this paper, we investigate how Monte Carlo can be used for diagonally dominant and some general matrices via a general splitting and how efficient mixed (stochastic/deterministic) parallel algorithms can be derived for obtaining an accurate inversion of a given non-singular matrix A. We employ either uniform Monte Carlo (UM) or almost optimal Monte Carlo (MAO) methods [7, 8, 9, 10, 11]. The relevant experiments with dense and sparse matrices are carried out.

Note that the algorithms are built under the requirement $\|T\| < 1$. Therefore to develop efficient methods we need to be able to solve problems with matrix norms greater than one. Thus we developed a spectrum of algorithms for MI and solving SLAEs ranging from special cases to the general case. Parallel MC methods for SLAEs based on Monte Carlo Jacobi iteration have been presented by Dimov [11]. Parallel Monte Carlo methods using minimum Makrov Chains and minimum communications are presented in [5, 1]. Most of the above approaches are based on the idea of balancing the stochastic and systematic errors [11]. In this paper we go a step further and have designed hybrid algorithms for MI and solving SLAEs by combining two ideas: iterative Monte Carlo methods based on the Jacobi iteration and deterministic procedures for improving the accuracy of the MI or the solution vector of SLAEs.

The generic Monte Carlo ideas are presented in Section 2, the main algorithms are described in Section 3 and the parallel approach and some numerical experiments are presented in Section 4 and 5 respectively.

2 Monte Carlo and Matrix Computation

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

$$Ax = b \tag{1}$$

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 $b = (b_1, b_2, ..., b_n)^t$.

Assume the general case $\|A\| > 1$. We consider the splitting $A = D - C$, where off-diagonal elements of D are the same as those of A, and the diagonal elements of D are defined as $d_{ii} = a_{ii} + \gamma_i ||A||$, choosing in most cases $\gamma_i > 1, i = 1, 2, ..., n$. We further consider $D = B - B_1$ where B is the diagonal matrix of D, e.g. $b_{ii} = d_{ii} i = 1, 2, ..., n$. As shown in [1] we could transform the system (1) to

$$x = Tx + f, \tag{2}$$

where $T = D^{-1}C$ and $f = D^{-1}b$. The multipliers γ_i are chosen so that, if it is possible, they reduce the norm of T to be less than 1 and reduce the number of Markov chains required to reach a given precision. We consider two possibilities, first, finding the solution of $x = Tx + f$ using Monte Carlo (MC) method if $\|T\| < 1$ or finding D^{-1} using MC and after that finding A^{-1}. Then, if required, obtaining the solution vector is found by $x = A^{-1}b$. Following the Monte Carlo method described in [7, 11] we define a Markov chain and weights $W_0 = 1, W_j = W_{j-1} \frac{T_{s_{j-1}s_j}}{p_{s_{j-1}s_{j-1}}}$ for $j = 1, 2, \cdots, n$. Consider now the random variable $\theta[g] = \frac{g_{s_0}}{p_{s_0}} \sum_{i=1}^{\infty} W_i f_{s_i}$. We use the following notation for the partial sum:

$$\theta_i[g] = \frac{g_{s_0}}{p_{s_0}} \sum_{j=0}^{i} W_j f_{s_j}. \tag{3}$$

Under condition $\|T\| < 1$, the corresponding Neumann series converges for any given f, and $E\theta_i[g]$ tends to (g, x) as $i \to \infty$. Thus, $\theta_i[g]$ can be considered as an estimate of (g, x) for i sufficiently large. To find an arbitrary component of the solution, for example, the r^{th} component of x, we should choose, $g = e(r) = (\underbrace{0, ..., 1}_{r}, 0, ..., 0)$ such that

$$e(r)_\alpha = \delta_{r\alpha} = \begin{cases} 1 & \text{i}f \quad r = \alpha \\ 0 & otherwise \end{cases} \tag{4}$$

It follows that $(g, x) = \sum_{\alpha=1}^{n} e(r)_\alpha x_\alpha = x_r$.

The corresponding Monte Carlo method is given by:

$$x_r = \hat{\Theta} = \frac{1}{N} \sum_{s=1}^{N} \theta_i[e(r)]_s,$$

where N is the number of chains and $\theta_i[e(r)]_s$ is the approximate value of x_r in the s^{th} chain. It means that using Monte Carlo method, we can estimate only one, few or all elements of the solution vector. We consider Monte Carlo with uniform transition probability (UM) $p_{\alpha\beta} = \frac{1}{n}$ and Almost optimal Monte Carlo method (MAO) with $p_{\alpha\beta} = \frac{|T_{\alpha\beta}|}{\sum_{\beta=1}^{n} |T_{\alpha\beta}|}$, where $\alpha, \beta = 1, 2, \ldots, n$. Monte Carlo MI is obtained in a similar way [3].

To find the inverse $A^{-1} = C = \{c_{rr'}\}_{r,r'=1}^{n}$ of some matrix A, we must first compute the elements of matrix $M = I - A$, where I is the identity matrix. Clearly, the inverse matrix is given by $C = \sum_{i=0}^{\infty} M^i$, which converges if $\|M\| < 1$.

To estimate the element $c_{rr'}$ of the inverse matrix C, we let the vector f be the following unit vector $f_{r'} = e(r')$.

We then can use the following Monte Carlo method for calculating elements of the inverse matrix C:

$$c_{rr'} \approx \frac{1}{N} \sum_{s=1}^{N} \left[\sum_{(j|s_j=r')} W_j \right], \tag{5}$$

where $(j|s_j = r')$ means that only

$$W_j = \frac{M_{rs_1} M_{s_1 s_2} \dots M_{s_{j-1} s_j}}{p_{rs_1} p_{s_1 s_2} \dots p_{s_{j-1} p_j}} \quad (6)$$

for which $s_j = r'$ are included in the sum (5).

Since W_j is included only into the corresponding sum for $r' = 1, 2, \dots, n$, then the same set of N chains can be used to compute a single row of the inverse matrix, which is one of the inherent properties of MC making them suitable for parallelization.

3 The Hybrid MC Algorithm

The basic idea is to use MC to find the approximate inverse of matrix D, refine the inverse (filter) and find A^{-1}. In general, we follow the approach described in [7, 11] We can then find the solution vector through A^{-1}. According to the general definition of a regular splitting [2], if A, M and N are three given matrices satisfying $A = M - N$, then the pair of matrices M, N are called regular splitting of A, if M is nonsingular and M^{-1} and N are non-negative.

Therefore, let A be a nonsingular diagonal dominant matrix. If we find a regular splitting of A such that $A = D - C$, the SLAE $x^{(k+1)} = Tx^{(k)} + f$, where $T = D^{-1}C$, and $f = D^{-1}b$ converges to the unique solution x^* if and only if $\|T\| < 1$ [2].

The efficiency of inverting diagonally dominant matrices is an important part of the process enabling MC to be applied to diagonally dominant and some general matrices. Consider now the algorithm which can be used for the inversion of a general non-singular matrix A. Note that in some cases to obtain a very accurate inversion of matrix D some filter procedures can be applied.

Algorithm1: Finding A^{-1}.

1. **Initial data:** Input matrix A, parameters γ and ϵ.
2. **Preprocessing:**
   - 2.1 **Split** $A = D - (D - A)$, where D is a diagonally dominant matrix.
   - 2.2 **Set** $D = B - B_1$ where B is a diagonal matrix $b_{ii} = d_{ii}$ $i = 1, 2, ..., n$.
   - 2.3 **Compute** the matrix $T = B^{-1}B_1$.

```
    2.4 **Compute** $||T||$, the Number of Markov Chains $N = (\frac{0.6745}{\epsilon} . \frac{1}{(1-||T||)})^2$.
3. **For** i=1 to n;
    3.1 **For** j=1 to j=N;
            **Markov Chain Monte Carlo Computation:**
        3.1.1 **Set** $t_k = 0$(stopping rule), $W_0 = 1$, $SUM[i] = 0$ and $Point = i$.
        3.1.2 **Generate** an uniformly distributed random number $nextpoint$.
        3.1.3 **If** $T[point][netxpoint]! = 0$.
        **LOOP**
            3.1.3.1 **Compute** $W_j = W_{j-1} \frac{T[point][netxpoint]}{P[point][netxpoint]}$.
            3.1.3.2 **Set** $Point = nextpoint$ and $SUM[i] = SUM[i] + W_j$.
            3.1.3.3 **If** $|W_j| < \gamma$, $t_k = t_k + 1$
            3.1.3.4 **If** $t_k \geq n$, end LOOP.
        3.1.4 **End If**
        3.1.5 **Else** go to step 3.1.2.
    3.2 **End of loop j**.
    3.3 **Compute** the average of results.
4. **End of loop i**.
5. **Obtain** The matrix $V = (I - T)^{-1}$.
6. **Therefore** $D^{-1} = VB^{-1}$.
7. **Compute** the MC inversion $D^{-1} = B(I - T)^{-1}$.
8. **Set** $D_0 = D^{-1}$ (approximate inversion) and $R_0 = I - DD_0$.
9. **use filter procedure** $R_i = I - DD_i$, $D_i = D_{i-1}(I + R_{i-1})$, $i = 1, 2, ..., m$, where $m \leq k$.
10. **Consider the accurate inversion of D** by step 9 given by $D_0 = D_k$.
11. **Compute** $S = D - A$ where $S$ can be any matrix with all non-zero elements in diagonal and all of its off-diagonal elements are zero.
12. **Main function** for obtaining the inversion of A based on $D^{-1}$ step 9:
    12.1 **Compute** the matrices $S_i, i = 1, 2, ..., k$, where each $S_i$ has just one element of matrix $S$.
    12.2 **Set** $A_0 = D_0$ and $A_k = A + S$
    12.3 **Apply** $A_k^{-1} = A_{k+1}^{-1} + \frac{A_{k+1}^{-1} S_{i+1} A_{k+1}^{-1}}{1 - trace(A_{k+1}^{-1} S_{i+1})}$, $i = k-1, k-2, ..., 1, 0$.
13. **Print**the inversion of matrix A.
14. **End** of algorithm.
```

4 Parallel Implementation

We have implemented the algorithm proposed on a cluster of PCs and an IBM SP3 machine under MPI. We have applied master/slave approach and we have run also on a miniGrid incorporating both the cluster and the SP3 machine.

Inherently, Monte Carlo methods for solving SLAE allow us to have minimal communication, i.e. to partition the matrix A, pass the non-zero elements of the dense (sparse) matrix to every processor, to run the algorithm in parallel on each processor computing $\lceil n/p \rceil$ rows (components) of MI or 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 A^{-1} or x. The splitting procedure and refinement are also parallelised and integrated in the parallel implementation. Even in the case, when we compute only k components $(1 \leq k \leq n)$ of the MI (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.

In addition an iterative filter process is used to improve the accuracy of the Markov Chain Monte Carlo calculated inverse. The iterative filter process is initialised by setting $D_0 = D^{-1}$ (where D^{-1} is the inverse from the Monte Carlo calculations). Then iteratively $R_i = I - DD_i$ and $D_{i+1} = D_i(I + R_i)$. These iterations continue $(i = 1, 2, \dots)$ until $\|R_i\| < \gamma$.

The data dependency graph for the iterative filter (Figure 1) directs us to the method for parallelising the iterative filter. Each iteration is calculated separately, with a master process coordinating the calculations slaves and deciding on whether further iterations are required.

The master starts an iteration by sending D_i to each of the slaves. Each of the slaves calculates: n/p columns of R_i; the partial row sums of R_i (required for calculating $\|R_i\|$); and n/p columns of D_{i+1}. The slaves then send the columns of D_{i+1} and the partial row sums of R_i to the master, which calculates $\|R_i\|$ and so decides if the inverse, D_{i+1} is of the required accuracy.

In this way we can obtain for the parallel time complexity of the MC procedure of the algorithm $O(nNL/p)$ where N denotes the number of Markov Chains. According to central limit theorem for the given error ϵ we have $N \geq \left(\frac{0.6745}{\epsilon \times (1 - \|T\|)}\right)^2$, L denotes the length of the Markov chains and L$\leq \left(\frac{\log(\gamma)}{\log\|T\|}\right)$,

$$\begin{tikzcd}
 & D_0 \arrow[d] \arrow[dr] & & & & \\
\|R_0\| & R_0 \arrow[l] \arrow[r] & D_1 \arrow[d] \arrow[dr] & & & \\
 & \|R_1\| & R_1 \arrow[l] \arrow[r] & D_2 \arrow[d] \arrow[dr] & & \\
 & & \|R_2\| & R_2 \arrow[l] \arrow[r] & D_3 \arrow[d] \arrow[dr] & \\
 & & & & &
\end{tikzcd}$$

Figure 1: Data Dependency Graph

where ϵ, γ show the accuracy of Monte Carlo approximation [3]. Parameters ϵ, γ are used for the stochastic and systematic error. Note that if rough approximations of the MI or solution of the SLAE is required we need to run only the Monte Carlo component of the algorithm. If a higher precision is required we need to also run the filter procedures, which will add complexity of $O(n^3/p)$ in case of sparce matrices for example. The absolute error of the solution for matrix inversion is $\left\| I - \hat{A}^{-1}A \right\|$, where A is the matrix whose inversion has to be found, and $\hat{A}^{-1}$ is the approximate MI. The computational time is shown in seconds.

5 Experimental Results

The algorithms run on partition of a 32 processor IBM SP3 machine and a workstation cluster with a 100 Mbps Ethernet network. Each workstation had an Intel Pentium III processor with 256 MB RAM and a 30 GB harddisk. Each workstation was running SUSE Linux 8.1. The MPI environment used was LAM MPI 7.0.

We have carried test with low precision $10^{-1} - 10^{-2}$ and higher precision $10^{-5} - 10^{-6}$ in order to investigate the balance between stochastic and de-

Matrix Size	Time (Dense Case) in seconds			
	4 proc.	8 proc.	12 proc.	16 proc.
250	59.269	24.795	16.750	14.179
500	329.072	177.016	146.795	122.622
1000	1840.751	989.423	724.819	623.087

Table 1: MC with filter procedures on the cluster

Matrix Size	Time (MC, Dense Case) in seconds	
	16 proc. (4 SP and 12 cluster)	16 proc. (8 SP and 8 cluster)
250	729.208	333.418
500	4189.225	1945.454

Table 2: MC with filter procedures on the miniGrid

terministic components of the algorithms based on the principle of balancing of errors (e.g. keeping the stochastic and systematic error of the same order) [7]. We have also compared the efficiency of parallel Monte Carlo and Quasi-Monte Carlo methods for solving SLAEs

Our results show that all the algorithms scale very well. The results show that if we need to refine the results using filter procedure the proportion of

Matrix Size	Time (Dense Case) in seconds			
	4 proc.	8 proc.	12 proc.	16 proc.
MC	48.819	20.909	13.339	9.691
QMC	0.744	0.372	0.248	0.186

Table 3: MC vs QMC without filtering on the cluster (matrix size 250 by 250)

the filter procedure time grows with the growth of the matrix size, so we need to limit these procedures if possible. The last table shows that Quasi-Monte Carlo is faster in finding rough approximation of the solution of SLAE. The second table shows that is important to balance computations in a Grid environment and communicate with larger chunks of data. For example in this case this can lead to more than twice reducing the computational time on the same number of processors.

6 Conclusion

In this paper we have introduced a hybrid Monte Carlo/deterministic algorithms for Matrix Computation for any non-singular matrix. We have compared the efficiency of the algorithm on a cluster of workstations and in a Grid environment. The results show that the algorithms scale very well in such setting, but a careful balance of computation should be maintained. Further experiments are required to determine the optimal number of chains required for Monte Carlo procedures and how best to tailor together Monte Carlo and deterministic refinement procedures.

References

[1] B. Fathi, B.Liu and V. Alexandrov, *Mixed Monte Carlo Parallel Algorithms for Matrix Computation* , Lecture Notes in Computer Science, No 2330, Springer-Verlag, 2002, pp 609-618

[2] Ortega, J., *Numerical Analysis*, SIAM edition, USA, 1990.

[3] Alexandrov V.N., *Efficient parallel Monte Carlo Methods for Matrix Computation,* Mathematics and computers in Simulation, Elsevier **47** pp. 113-122, Netherlands, (1998).

[4] Golub, G.H., Ch., F., Van Loan, *Matrix Computations,* The Johns Hopkins Univ. Press, Baltimore and London, (1996)

[5] Taft K. and Fathi Vajargah B., *Monte Carlo Method for Solving Systems of Linear Algebraic Equations with Minimum Markov Chains.* International Conference PDPTA'2000 Las Vegas, (2000).

[6] Sobol I.M. *Monte Carlo Numerical Methods.* Moscow, Nauka, 1973 (in Russian).

[7] Dimov I., Alexandrov V.N. and Karaivanova A., *Resolvent Monte Carlo Methods for Linear Algebra Problems,* Mathematics and Computers in Simulation, Vo155, pp. 25-36, 2001.

[8] Fathi Vajargah B. and Alexandrov V.N., *Coarse Grained Parallel Monte Carlo Algorithms for Solving Systems of Linear Equations with Minimum Communication,* in Proc. of PDPTA, June 2001, Las Vegas, 2001, pp. 2240-2245.

[9] Alexandrov V.N. and Karaivanova A., *Parallel Monte Carlo Algorithms for Sparse SLAE using MPI,* LNCS 1697, Springer 1999, pp. 283-290.

[10] Alexandrov V.N., Rau-Chaplin A., Dehne F. and Taft K., *Efficient Coarse Grain Monte Carlo Algorithms for matrix computation using PVM,* LNCS 1497, pp. 323-330, Springer, August 1998.

[11] Dimov I.T., Dimov T.T., et all, *A new iterative Monte Carlo Approach for Inverse Matrix Problem,* J. of Computational and Applied Mathematics **92** pp 15-35 (1998).