Mixed Monte Carlo Parallel Algorithms for Matrix Computation

Behrouz Fathi, Bo Liu, and Vassil Alexandrov

Department of Computer Science,
University of Reading,
Whiteknights P.O. Box 225
United Kingdom, RG6 6AY
{b.fathivajargah, b.liu, v.n.alexandrov}@rdg.ac.uk

Abstract. In this paper we consider mixed (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. In this paper 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. Experimental results with dense and sparse matrices are presented.

Keywords: Monte Carlo Method, Markov Chain, Matrix Inversion, Solution of sytem of Linear Equations, Matrix Decomposition, Diagonal Dominant Matrices.

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.

P.M.A. Sloot et al. (Eds.): ICCS 2002, LNCS 2330, pp. 609–618, 2002.

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 [7–11]. 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 [7–11]. 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 between master/slave processors 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 mixed 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 $x = A^{-1}b$.

Consider first the stochastic approach. Assume that $\|T\| < 1$ and that the system is transformed to its iterative form (2). Consider the Markov chain given by:

$$s_0 \to s_1 \to \cdots \to s_k, \tag{3}$$

where the $s_i, i = 1, 2, \cdots, k$, belongs to the state space $S = \{1, 2, \cdots, n\}$. Then for $\alpha, \beta \in S, p_0(\alpha) = p(s_0 = \alpha)$ is the probability that the Markov chain starts at state α and $p(s_{j+1} = \beta | s_j = \alpha) = p_{\alpha\beta}$ is the transition probability from state α to state β. The set of all probabilities $p_{\alpha\beta}$ defines a transition probability matrix $P = \{p_{\alpha\beta}\}_{\alpha,\beta=1}^{n}$ [3, 9, 10]. We say that the distribution $(p_1, \cdots, p_n)^t$ is acceptable for a given vector g, and that the distribution $p_{\alpha\beta}$ is acceptable for matrix T, if $p_\alpha > 0$ when $g_\alpha \neq 0$, and $p_\alpha \geq 0$, when $g_\alpha = 0$, and $p_{\alpha\beta} > 0$ when $T_{\alpha\beta} \neq 0$, and $p_{\alpha\beta} \geq 0$ when $T_{\alpha\beta} = 0$ respectively. We assume $\sum_{\beta=1}^{n} p_{\alpha\beta} = 1$, for all $\alpha = 1, 2, \cdots, n$. Generally, we define

$$W_0 = 1, W_j = W_{j-1} \frac{T_{s_{j-1}s_j}}{p_{s_{j-1}s_{j-1}}} \tag{4}$$

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:

$$\theta_i[g] = \frac{g_{s_0}}{p_{s_0}} \sum_{j=0}^{i} W_j f_{s_j}. \tag{5}$$

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

$$e(r)_\alpha = \delta_{r\alpha} = \begin{cases} 1 & if \quad r = \alpha \\ 0 & otherwise \end{cases} \tag{6}$$

It follows that

$$(g, x) = \sum_{\alpha=1}^{n} e(r)_\alpha x_\alpha = x_r. \tag{7}$$

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

$$C = \sum_{i=0}^{\infty} M^i, \tag{8}$$

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

$$f_{r'} = e(r'). \tag{9}$$

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

$$c_{rr'} \approx \frac{1}{N}\sum_{s=1}^{N}\left[\sum_{(j|s_j=r')} W_j\right], \tag{10}$$

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

$$W_j = \frac{M_{rs_1}M_{s_1s_2}\ldots M_{s_{j-1}s_j}}{p_{rs_1}p_{s_1s_2}\cdots p_{s_{j-1}p_j}} \tag{11}$$

for which $s_j = r'$ are included in the sum (10).

Since W_j is included only into the corresponding sum for $r' = 1, 2, \ldots, n$, 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.

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 $\bar{\theta}$ [6].

3 The Mixed 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}. 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 as $A = D - C$ than the SLAE $x^{(k+1)} = Tx^{(k)} + f$, where $T = D^{-1}C$, and $f = D^{-1}b$ converge to 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. The basic algorithms, covering the inversion of diagonally dominant and arbitrary non-singular matrix are given below. First let us consider how we can find Monte Carlo approximate of D^{-1} :

Algorithm1: Finding D^{-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 $N = (\frac{0.6745}{\epsilon} \cdot \frac{1}{(1-||T||}$)^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. **End** of algorithm.

Consider now the second 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.

Algorithm2: 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 $N = (\frac{0.6745}{\epsilon} \cdot \frac{1}{(1-||T||}$)^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 algorithms proposed on a cluster of workstations and a Silicon Graphics ONYX2 machine under PVM. We have applied master/slave approach.

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 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 $\mathrm{L} \leq \left(\frac{\log(\gamma)}{\log\|T\|}\right)$, where ϵ, γ show the accuracy of Monte Carlo approximation [3]. Parameters ϵ, γ are used for the stochastic and systematic error. 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 a 12 processor (four 195 MHZ, eight 400MHZ) $ONYX2$ Silicon Graphics machine with 5120 Mbytes main memory. 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 deterministic 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].

Consider now, firstly, a low precision 10^{-1}, approximation to MI for randomly generated dense and sparse matrices and secondly higher precision 10^{-5} approximation to MI for randomly generated dense and sparse matrices.

Matrix Size	Dense Case		
	Time(MC)	Time(total)	Error
100	0.51901	1.135063	0.1
300	2.11397	5.32793	0.1
500	12.7098	69.34915	0.1
1000	59.323708	199.60263	0.1
2000	338.4567	1189.3193	0.1
3000	437.072510	2295.97385	0.1
4000	793.01563	3321.87578	0.1
5000	1033.9867	3979.39237	0.1
6000	1265.5816	5162.56743	0.1
7000	1340.0085	5715.66747	0.1
8000	1645.2306	5991.52939	0.1

Table 1. Dense randomly generated matrices

Matrix Size	Sparse Case		
	Time(MCMC)	Time(total)	Error
100	0.238264	1.127853	0.1
300	0.85422	3.27931	0.1
500	7.95166	29.11630	0.1
1000	47.5126	157.6298	0.1
2000	315.5001	733.28491	0.1
3000	389.2388	1466.9328	0.1
4000	696.9654	1994.95163	0.1
5000	819.3842	2553.76525	0.1
6000	994.8456	3413.86492	0.1
7000	1143.4664	3759.7539	0.1
8000	1389.7674	3955.664901	0.1

Table 2. Sparse randomly generated matrices

Matrix Size	Dense Case	
	Time(total)	Error
100	2.14282	0.002121
300	11.00373	0.000626
500	118.29381	0.000398
1000	799.9064	0.000266
2000	4989.2433	0.000044
3000	5898.4772	0.000038
4000	6723.6863	0.000027
5000	7159.7639	0.000015
6000	7695.0903	0.000006
7000	8023.7381	0.000003
8000	8914.5512	0.00000018

Table 3. Higher precision MI for dense matrices

Matrix Size	Sparse Case	
	Time(total)	Error
100	1.9232	0.000611000
300	8.14498	0.000216000
500	90.25106	0.000163000
1000	763.32165	0.000078000
2000	4758.98103	0.000055000
3000	5678.2463	0.000018000
4000	6318.9001	0.000007200
5000	6855.4118	0.000001400
6000	7057.8491	0.000000510
7000	7528.5516	0.000000220
8000	8018.921	0.000000050

Table 4. Higher precision MI for sparse matrices

6 Conclusion

In this paper we have introduced a mixed Monte Carlo/deterministic algorithms for Matrix Inversion and finding a solution to SLAEs for any non-singular matrix. 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. We have shown also that the algorithms run efficiently in parallel and the results confirmed their efficiency for MI with dense and sparse matrices. Once good enough inverse is found, we can find the solution vector of SLAE. The accuracy of results are comparable with other known computational algorithms.

References

1. Fathi Vajargah B., Liu B. and Alexandrov V., On the preconditioner Monte Carlo methods for solving linear systems. MCM 2001, Salzburg, Austria (presented).
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).