Using P-GRADE for Monte Carlo Computations in a Distributed Environment

Vassil N. Alexandrov[1], Ashish Thandavan[1], and Péter Kacsuk[2]

[1] Department of Computer Science, University of Reading, Reading, UK
[2] MTA SZTAKI Research Institute, Budapest, Hungary

Abstract. Computations involving Monte Carlo methods are, very often, easily and efficiently parallelized. P-GRADE is a parallel application development environment which provides an integrated set of programming tools for development of general message-passing applications to run in heterogeneous computing environments or supercomputers. In this paper, we show how Monte Carlo algorithms for solving Systems of Linear Equations and Matrix Inversion can easily be parallelized using P-GRADE.

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].

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 L is an estimate of the chain length in the stochastic process, which are independent of matrix size n and secondly, these stochastic methods are inherently parallel.

Several authors have proposed different coarse grained Monte Carlo parallel algorithms for MI and SLAE [6,7,8,9,10]. 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 [6,7,8,9,10].

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

M. Bubak et al. (Eds.): ICCS 2004, LNCS 3039, pp. 475–482, 2004.

by Dimov [10]. Parallel Monte Carlo methods using minimum Makrov Chains and minimum communications are presented in [1]. Most of the above approaches are based on the idea of balancing the stochastic and systematic errors [10]. In this paper we have presented a 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 in Sections 2 and 3. Further the parallel approach using P-GRADE 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. In the general case we consider finding D^{-1} using MC and after that finding A^{-1}. Then, if required, 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 $P = \{p_{\alpha\beta}\}_{\alpha,\beta=1}^{n}$ [3,8,9]. 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 $e(r)_\alpha = \delta_{r\alpha} = \begin{cases} 1 & if \quad r = \alpha \\ 0 & otherwise \end{cases}$

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:

$$c_{rr'} \approx \frac{1}{N} \sum_{s=1}^{N} \left[\sum_{(j|s_j = r')} W_j \right], \tag{6}$$

where $(j|s_j = r')$ means that only W_j for which $s_j = r'$ are included in the sum.

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 $\bar{\theta}$ [5].

3 The Hybrid MC Algorithm

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.

Algorithm: 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.

The basic idea is to use MC to find the approximate inverse of matrix D, refine the inverse (filter) and find A^{-1}. According to the general definition of a regular splitting [2], if A, M and N are three given matrices and $A = M-N$, 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].

4 Parallelisation Using P-GRADE

The Parallel GRaphical Application Development Environment is, as the name suggests, a parallel programming environment which supports the whole life-cycle of parallel program development. All the stages from initial design to execution and debugging to performance visualisation and tuning of the parallel application are supported by P-GRADE.

It uses a combination of graphics and program statements to describe what the application does. The execution environment can be a varied one, ranging from clusters of workstations to supercomputers. A parallel application typically consists of two or more processes which communicate via messages. Two popular message passing libraries used for parallel programming are Parallel Virtual Machine (PVM) and Message Passing Interface (MPI). P-GRADE allows the developer to choose which library he / she wishes to use without needing to know the syntax of the underlying message passing system. All the messages are generated automatically from the graphics. Compilation and distribution of the executables are performed automatically in the heterogeneous environment. An integrated debugger allows the program to be methodically debugged during runtime and monitoring and visualisation tools provide performance information.

4.1 Tools in P-GRADE

P-GRADE consists of a few main components. The application developer uses the GRED [12] editor to design and construct the parallel program. The program flow is described by a special graphical programming language called GRAPNEL. The GRP2C precompiler compiles the graphical information into C code with PVM or MPI. It also creates additional makefiles which are used by the UNIX make utility to build the application executables.

Once the executables have been created, the parallel program can be executed either in debugging mode or in trace mode. In the debugging mode, the execution of the program is under the control of the DIWIDE [11] distributed debugger which provides options to create breakpoints, perform step-by-step execution, animation of the flow of control, etc. Once the program has been successfully debugged, it can be executed in trace mode. GRM [13], a distributed

monitoring tool, is responsible for generating a file containing trace events defined by the developer. The collected data can then be visualised by the PROVE [13] visualization tool, which is invaluable in assisting the developer to locate performance bottlenecks in the running application.

P-GRADE currently supports job execution in interactive as well as batch mode. An application could be executed interactively on a cluster of workstations the workstations involved are declared beforehand and processes are assigned to run on them by PVM or MPI. On the other hand, the application could be submitted as a job to a batch scheduling system like Condor, which would be responsible for assigning processes to resources. In future versions of P-GRADE, the target execution environment could be a computational grid managed by Globus.

4.2 Parallel 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 (or its partitions) 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.

In P-GRADE we employed a master/slave approach, where the main process had to read the data from a file, partition it, send it out to the slaves and collect the results from them at the end of the computation. The slaves were defined using the Process Farm template in P-GRADE which allows scaling to larger number of processes when more compute resources are available. The GRM and PROVE tools were extremely useful in fine-tuning the performance of the application.

5 Numerical Experiments

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

We have carried out tests 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) [6]. Consider now, finding the solution to SLAE using Monte Carlo and applying the filter procedures with precision $10^{-5} - 10^{-6}$:

Table 1. MC with filter procedures on the cluster

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 2. MC with filter procedures on the miniGrid

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

The above results show that all the algorithms scale very well. The second table shows that it 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 a substantial reduction of computational time.

6 Conclusion

In this paper we have considered how we can efficiently use P-GRADE for programming 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.

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. Sobol I.M. *Monte Carlo Numerical Methods.* Moscow, Nauka, 1973 (in Russian).
6. 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.
7. 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.
8. Alexandrov V.N. and Karaivanova A., *Parallel Monte Carlo Algorithms for Sparse SLAE using MPI,* LNCS 1697, Springer 1999, pp. 283-290.
9. 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.
10. 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).
11. Kacsuk P., Lovas R. and Kovács J., *Systematic Debugging of Parallel Programs in DIWIDE Based on Collective Breakpoints and Macrosteps*, Proc. of the 5th International Euro-Par Conference, Toulouse, France, 1999, pp. 90-97.
12. Kacsuk P., Dózsa G., Fadgyas T. and Lovas R. *The GRED Graphical Editor for the GRADE Parallel Program Development Environment*, Journal of Future Generation Computer Systems, Vol. 15(1999), No. 3, pp. 443-452.
13. Balaton Z., Kacsuk P. and Podhorszki N., *Application Monitoring in the Grid with GRM and PROVE* , Proc. of the International Conference on Computational Science, ICCS 2001, San Francisco, CA., USA. pp. 253-262.