Implementation of Monte Carlo Algorithms for Eigenvalue Problem Using MPI

I. Dimov[1], V. Alexandrov[2], and A. Karaivanova[1]

[1] Central Laboratory for Parallel Processing, Bulgarian Academy of Sciences,
Acad. G. Bonchev St.,bl. 25 A, 1113, Sofia, Bulgaria,
dimov@amigo.acad.bg, anet@copern.acad.bg,
WWW home page: http://www.acad.bg/BulRTD/math/dimov2.html
[2] Department of Computer Science University of Liverpool
Chadwick Building Peach Street, Liverpool, L69 7ZF, UK
vassil@csc.liv.ac.uk

Abstract. The problem of evaluating the dominant eigenvalue of real matrices using Monte Carlo numerical methods is considered.
Three almost optimal Monte Carlo algorithms are presented:

- Direct Monte Carlo algorithm (DMC) for calculating the largest eigenvalue of a matrix A. The algorithm uses iterations with the given matrix.
- Resolvent Monte Carlo algorithm (RMC) for calculating the smallest or the largest eigenvalue. The algorithm uses Monte Carlo iterations with the resolvent matrix and includes parameter controlling the rate of convergence;
- Inverse Monte Carlo algorithm (IMC) for calculating the smallest eigenvalue. The algorithm uses iterations with inverse matrix.

Numerical tests are performed for a number of large sparse test matrices using MPI on a cluster of workstations.

1 Introduction

Monte Carlo methods give statistical estimates for the functional of the solution by performing random sampling of a certain random variable whose mathematical expectation is the desired functional. They can be implemented on parallel machines efficiently due to their inherent parallelism and loose data dependencies. Using powerful parallel computers it is possible to apply Monte Carlo method for evaluating large-scale irregular problems which sometimes are difficult to be solved by the well-known numerical methods.

Let J be any functional that we estimate by Monte Carlo method; θ_N be the estimator, where N is the number of trials. The probable error for the usual Monte Carlo method [5] is defined as parameter r_N for which $Pr\{|J-\theta_N| \geq r_N\} = 1/2 = Pr\{|J-\theta_N| \leq r_N\}$. If the standard deviation is bounded, i.e. $D(\theta_N) < \infty$, the normal convergence in the central limit theorem holds, so we have

$$r_N \approx 0.6745 D(\theta_N) N^{-1/2}. \tag{1}$$

V. Alexandrov and J. Dongarra (Eds.): PVM/MPI'98, LNCS 1497, pp. 346–353, 1998.

In this paper we present Monte Carlo algorithms for evaluating the dominant eigenvalue of large real sparse matrices and their parallel implementation on a cluster of workstations using MPI. These three algorithms use the idea of the Power method combined with Monte Carlo iterations by the given matrix, the resolvent matrix and the inverse matrix correspondingly. In [6], [5], [3], [4] one can find Monte Carlo methods for evaluation the dominant (maximal by modulus) eigenvalue of an integral operator. In [7], [8] Monte Carlo algorithms for evaluating the smallest eigenvalue of real symmetric matrices are proposed. Here we generalize the problem.

Power method.

Suppose $A \in R^{n\times n}$ is diagonalizable, $X^{-1}AX = diag(\lambda_1, \ldots, \lambda_n)$, $X = (x_1, \ldots, x_n)$, and $|\lambda_1| > |\lambda_2| \geq \ldots \geq |\lambda_n|$. Given $f^{(0)} \in C^n$, the *power method* ([1]) produces a sequence of vectors $f^{(k)}$ as follows:

$z^{(k)} = Af^{(k-1)}$,
$f^{(k)} = z^{(k)}/||z^{(k)}||_2$,
$\lambda^{(k)} = [f^{(k)}]^H Af^{(k)}$, $k = 1, 2, \ldots$.

Except for special starting points, the iterations converge to an eigenvector corresponding to the eigenvalue of A with largest magnitude (*dominant eigenvalue*) with rate of convergence:

$$|\lambda_1 - \lambda^{(k)}| = O\left(\left|\frac{\lambda_2}{\lambda_1}\right|^k\right). \tag{2}$$

Consider the case when we want to compute the smallest eigenvalue. To handle this case and others, the power method is altered in the following way: The iteration matrix A is replaced by B, where A and B have the same eigenvectors, but different eigenvalues. Letting σ denote a scalar, then the three common choices for B are: $B = A - \sigma I$ which is called the *shifted power method*, $B = A^{-1}$ which is called the *inverse power method*, and $B = (A - \sigma I)^{-1}$ which is called the *inverse shifted power method*.

Table 1. Relationship between eigenvalues of A and B

B	Eigenvalue of B	Eigenvalue of A
A^{-1}	$\frac{1}{\lambda_A}$	$\frac{1}{\lambda_B}$
$A - \sigma I$	$\lambda_A - \sigma$	$\lambda_B + \sigma$
$(A - \sigma I)^{-1}$	$\frac{1}{\lambda_A - \sigma}$	$\sigma + \frac{1}{\lambda_B}$

Computational Complexity: Having k iterations, the number of arithmetic operations in Power method is $O(4kn^2 + 3kn)$, so the Power method is not

suitable for large sparse matrices. **In order to reduce the computational complexity we propose a Power method with Monte Carlo iterations**.

2 Monte Carlo Algorithms

2.1 Monte Carlo Iterations

Consider a matrix $A = \{a_{ij}\}_{i,j=1}^{n}, A \in R^{n\times n}$, and vectors $f = (f_1, \ldots, f_n)^t \in R^{n\times 1}$ and $h = (h_1, \ldots, h_n)^t \in R^{n\times 1}$. The algebraic transformation $Af \in R^{n\times 1}$ is called *iteration* and plays a fundamental role in iterative Monte Carlo methods.

Consider the following Markov chain:

$$k_0 \to k_1 \to \ldots \to k_i, \tag{3}$$

where $k_j = 1, 2, \ldots, n$ for $j = 1, \ldots, i$ are natural numbers. The rules for constructing the chain (3) are:

$$Pr(k_0 = \alpha) = \frac{|h_\alpha|}{\sum_{\alpha=1}^{n} |h_\alpha|},$$

$$Pr(k_j = \beta | k_{j-1} = \alpha) = \frac{|a_{\alpha\beta}|}{\sum_{\beta=1}^{n} |a_{\alpha\beta}|}, \ \alpha = 1, \ldots, n.$$

Such a choice of the initial density vector and the transition density matrix leads to *almost optimal* Monte Carlo algorithms for matrix computations.

Now define the random variables W_j using the following recursion formula:

$$W_0 = \frac{h_{k_0}}{p_{k_0}}, \ \ W_j = W_{j-1}\frac{a_{k_{j-1}k_j}}{p_{k_{j-1}k_j}}, \ \ j = 1, \ldots, i. \tag{4}$$

2.2 Direct Monte Carlo Algorithm

The dominant eigenvalue can be obtained using the iteration process mentioned in the Introduction:

$$\lambda_{max} = lim_{i\to\infty} \frac{(h, A^i f)}{(h, A^{i-1} f)},$$

where we calculate scalar products having in mind (see, [5], [6], [2]) that

$$(h, A^i f) = E\{W_i f_{k_i}\}, \ \ i = 1, 2, \ldots. \tag{5}$$

Thus we have

$$\lambda_{max} \approx \frac{E\{W_i f_{k_i}\}}{E\{W_{i-1} f_{k_{i-1}}\}}$$

2.3 Inverse Shifted Monte Carlo Algorithm (Resolvent MC Method)

Now consider an algorithm based on Monte Carlo iterations by the resolvent matrix $R_q = [I - qA]^{-1} \in R^{n \times n}$. The following presentation holds

$$[I - qA]^{-m} = \sum_{i=0}^{\infty} q^i C_{m+i-1}^i A^i, |qA| < 1.$$

Having in mind that (see, [3], [7])

$$([I - qA]^{-m} f, h) = E\left\{\sum_{i=0}^{\infty} q^i C_{m+i-1}^i (A^i f, h)\right\}, \tag{6}$$

we have the following Monte Carlo algorithm:

$$\lambda_{min} \approx \frac{1}{q}\left(1 - \frac{1}{\mu^{(m)}}\right) = \frac{(A[I - qA]^{-m} f, h)}{([I - qA]^{-m} f, h)} =$$

$$\frac{E\sum_{i=1}^{\infty} q^{i-1} C_{i+m-2}^{i-1} W_i f(x_i)}{E\sum_{i=0}^{\infty} q^i C_{i+m-1}^i W_i f(x_i)} = \frac{E\sum_{i=0}^{l} q^i C_{i+m-1}^i W_{i+1} f(x_{i+1})}{E\sum_{n=0}^{l} q^i C_{i+m-1}^i W_i f(x_i)}, \tag{7}$$

where $W_0 = \frac{h_{k_0}}{p_{k_0}}$, W_i are defined by (4) and C_i^j are binomial coefficients.

Let us note that if $q > 0$ the algorithm evaluates λ_{max}, if $q < 0$, the algorithm evaluates λ_{min} without matrix inversion. This parameter may be used as parameter controlling the convergency.

The coefficients C_{n+m}^n are calculated using the formula

$$C_{i+m}^i = C_{i+m-1}^i + C_{i+m-1}^{i-1}.$$

The RMC algorithm has strong requirements about matrices for which it can be applied: the error from the Power method applied on the resolvent matrix determines the value of the parameter m; the error which comes from the representation of the resolvent matrix as a series determines the value of the parameter l, and also the values of m and l are not independent, since they determine the binomial coefficients $C_{m=l-1}^l$ which grow exponentially with l.

2.4 Inverse Monte Carlo Algorithm

This algorithm can be applied when A is a non-singular matrix. The algorithm has high efficiency when the smallest by modulus eigenvalue of A is much smaller than the other eigenvalues. This algorithm can be implemented in two ways:

– **First**,
1. Calculate the inversion of matrix A. For example, an efficient algorithm for evaluating the inverse matrix is proposed in [9].
2. Apply the Direct Monte Carlo Algorithm using the iterations with the inverse matrix.
Remark. It is not allways necessary to calculate A^{-1} because the vectors f_k can be evaluated by solving the following systems of equations:

$$Af_j = f_{j-1}, j = 1, \ldots, i,$$

where $f_j = A^{-j} f_{j-1}$ and f_0 is the starting vector.
– **Second**, to apply Resolvent Monte Carlo algorithm with $q = -1$, i.e.

$$\lambda_{min} \approx \frac{E \sum_{i=0}^{l} C_{i+m-1}^{i} W_{i+1} f(x_{i+1})}{E \sum_{n=0}^{l} C_{i+m-1}^{i} W_i f(x_i)}$$

2.5 Balancing of Errors

There are two kind of errors in Power method with Monte Carlo iterations:

– systematic error (from Power method, see (2)):

$$O\left(\left|\frac{\mu_2}{\mu_1}\right|^k\right),$$

where $\mu_i = \lambda_i$ if $B = A$, $\mu_i = \frac{1}{\lambda_i}$ if $B = A^{-1}$, $\mu_i = \lambda_i - \sigma$ if $B = A - \sigma I$, $\mu_i = \frac{1}{\lambda_i - \sigma}$ if $B = (A - \sigma I)^{-1}$ and λ_i and μ_i are the eigenvalues of A and B correspondingly;
– stochastic error (because we calculate mathematical expectations approximately, see (1)):

$$O(D(\theta_N) N^{-1/2})$$

To obtain good results the stochastic error must be approximately equal to the systematic one. (It is not necessary to use "a large" number of realizations N in order to have "a small" stochastic error if the systematic error is "big".)

2.6 Computational Complexity

The mathematical expectation of the total number of operations for the Resolvent MC Method ([7], [8] is:

$$ET_1(RMC) \approx 2\tau \left[(k + \gamma_A) l_A + \frac{1}{2} d l_L\right] lN + 2\tau n(1 + d), \tag{8}$$

where l is the numbers of moves in every Markov chain, N is the number of Markov chains, d is the mean value of the number of non-zero elements per row, γ_A is the number of arithmetic operations for calculation the random variable

(in our code-realization of the algorithm $\gamma_A = 6$), l_A and l_L are arithmetic and logical suboperations in one move of the Markov chain, k is the number of arithmetic operations for generating the random number (k is equal to 2 or 3).

The main term of (8) does not depend on the matrix size n. This means that the time required for calculating the eigenvalue by RMC practically does not depend n. The parameters l and N depend on the spectrum of the matrix, but does not depend on its size n. The above mentioned result was confirmed for a wide range of matrices during the realized numerical experiments (see Table 3).

3 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. Each processor executes the same program for N/p number of trajectories, i.e. it computes N/p independent realizations of the random variable (here p is the number of processors). 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.

The test matrices are sparse and storaged in *packed row format* (i.e. only nonzero elements). The results for average time and efficiency are given in tables 2 and 3 and look promising. The relative accuracy is 10^{-3}.

We consider the *parallel efficiency* E as a measure that characterize the quality of the proposed algorithms. We use the following definition:

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

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

4 Conclusion

Parallel Monte Carlo algorithms for calculating the eigenvalues are presented and studied. They can be applied for well balanced matrices (which have nearly equal sums of elements per row) in order to provide good accuracy.

We propose to use them when one have to calculate the dominant eigenvalue of very large sparse matrices since the computational time is almost independent of the dimension of the matrix and their parallel efficiency is superlinear.

5 Acknowledgements

This work was partially supported by the Ministry of Science, education and Technology of Bulgaria under grant I501/95 and by the European Community

Table 2. Implementation of **Direct Monte Carlo Algorithm** using MPI (number of trajectories - 100000).

	$1pr.$ $T(ms)$	$2pr.$ $T(ms)$	$2pr.$ E	$3pr.$ $T(ms)$	$3pr.$ E	$4pr.$ $T(ms)$	$4pr.$ E	$5pr.$ $T(ms)$	$5pr.$ E
matrix n = 128	34	17	1	11	1.03	8	1.06	7	0.97
matrix n = 1024	111	56	0.99	37	1	27	1.003	21	1.06
matrix n = 2000	167	83	1	56	1	42	1	35	0.96

Table 3. Implementation of **Resolvent Monte Carlo Algorithm** for evaluation of λ_{max} using MPI (number of trajectories - 100000; $q > 0$).

	$1pr.$ $T(ms)$	$2pr.$ $T(ms)$	$2pr.$ E	$3pr.$ $T(ms)$	$3pr.$ E	$4pr.$ $T(ms)$	$4pr.$ E	$5pr.$ $T(ms)$	$5pr.$ E
matrix n = 128	18	9	1	6	1	4	1.1	3	1.2
matrix n = 1024	30	15	1	10	1	7	1.06	6	1
matrix n = 2000	21	11	0.99	7	1	5	1.04	4	1.04

grant CP960237 (Study of Stability of Physical Systems Using Parallel Computers).

References

1. G. H. Golub, Ch. F. Van Loon, *Matrix Computations*, **The Johns Hopkins Univ. Press**, Baltimore and London, 1996.
2. 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.
3. G.A. Mikhailov, *A new Monte Carlo algorithm for estimating the maximum eigenvalue of an integral operator*, **Docl. Acad. Nauk SSSR, 191**, No 5 (1970), pp. 993 – 996.
4. G.A. Mikhailov, Optimization of the "weight" Monte Carlo methods (Nauka, Moscow, 1987).
5. I.M. Sobol, *Monte Carlo numerical methods*, **Nauka**, Moscow, 1973.

6. V.S.Vladimirov, *On the application of the Monte Carlo method to the finding of the least eigenvalue, and the corresponding eigenfunction*, of a linear integral equation, in Russian: **Teoriya Veroyatnostej i Yeye Primenenie, 1**, No 1 (1956),pp. 113 – 130.
7. I. Dimov, A. Karaivanova and P. Yordanova, *Monte Carlo Algorithms for calculating eigenvalues*, **Springer Lectur Notes in Statistics, v.127** (1998) (H. Niederreiter, P. Hellekalek, G. Larcher and P. Zinterhof, Eds)), pp.205-220.
8. I. Dimov, A. Karaivanova *Parallel computations of eigenvalues based on a Monte Carlo approach*, **Journal of MC Methods and Appl.**, 1998 (to appear).
9. Megson, G., V. Alexandrov, I. Dimov, *Systolic Matrix Inversion Using a Monte Carlo Method*, **Journal of Parallel Algorithms and Applications** , 3, No 1 (1994), pp. 311-330.