Parallel resolvent Monte Carlo algorithms for linear algebra problems

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

[a] *CLPP, Bulgarian Academy of Sciences, Acad. G. Bonchev str., bl. 25 A, 1113 Sofia, Bulgaria*

[b] *Department of Computer Science, University of Liverpool, Liverpool, UK*

Abstract

In this paper we consider Monte Carlo (MC) algorithms based on the use of the resolvent matrix for solving linear algebraic problems. Estimates for the speedup and efficiency of the algorithms are presented. Some numerical examples performed on cluster of workstations using MPI are given.

1 Introduction

The goals of this work are:

- to develop a common Monte Carlo (MC) numerical approach based on the use of resolvent matrices for solving linear algebra problems;
- to show that the proposed resolvent approach improves the convergence of the algorithms;
- to realize parallel versions of resolvent MC algorithms using MPI and to present numerical results showing that they have the same nice parallel properties as the simple iterative MC.

We consider MC numerical algorithms and their parallel realization for the following algebraic problems:

[1] Partially supported by the Ministry of Science and Education of Bulgaria under grant I-811/1998.

(a) Evaluating the inner product

$$J(u) = (h, u) = \sum_{i=1}^{n} h_i u_i \tag{1}$$

for the solution $u \in I\!\!R^{n\times 1}$ of the linear algebraic system $Au = b$, where $A = \{a_{ij}\}_{i,j=1}^{n} \in I\!\!R^{n\times n}$ is a given *sparse* matrix; $b = (b_1, \ldots, b_n)^t \in I\!\!R^{n\times 1}$ and $h = (h_1, \ldots, h_n)^t \in I\!\!R^{n\times 1}$ are given vectors.

(b) Finding one or more eigenvalues λ such that:

$$Ax = \lambda x, \tag{2}$$

where $A = \{a_{ij}\}_{i,j=1}^{n} \in I\!\!R^{n\times n}$ is a given *sparse* matrix. This is a special case of (2) where the right-hand-side vector is a (real or complex) scalar multiple of the unknown vector x.

In case **(a)** once we can establish that a unique solution to the given system of equations exists, we choose a non-singular matrix $M \in I\!\!R^{n\times n}$ such that $MA = I - L$, where $I \in R^{n\times n}$ is the identity matrix and $Mb = f$, $f \in I\!\!R^{n\times 1}$. Then

$$u = Lu + f \tag{3}$$

and we can consider the iteration process

$$u_k = L(u_{k-1}) + f, \;\; k = 1, 2, \ldots \tag{4}$$

Under the usual conditions:

$$\textbf{(i)} \qquad \begin{cases} 1.\ \text{The matrices } M \text{ and } L \text{ are both non-singular;} \\ 2.\ |\lambda(L)| < 1 \;\text{ for all eigenvalues }\; \lambda(L) \;\text{ of }\; L, \end{cases}$$

the iteration (4) converges to the solution with the rate: $\| \; u_k - u \; \| \leq \| \; L \; \|^k \| \; u_0 - u) \; \|$.

There are many classical numerical methods for solving the linear systems of equations of the general form $AX = B$, where the $(n \times n)$ matrix A and the $(n \times m)$ matrix B are known, while the $(n \times m)$ matrix X is the unknown quantity to be determined. The *direct methods*, such as *Gaussian* elimination, and LU and *Cholesky decomposition* techniques, take time

$$T_{DIRECT} = O(n^3) + O(n^2 m);$$

while the *iterative methods*, such as the *Jacobi, Gauss- Seidel*, and various *relaxation* techniques, take time

$$T_{ITER} = O(n^2ml),$$

if there are l iterations. Even if l and m are relatively small, this becomes too laborious if n is large.

For the same problem *Monte Carlo* sequential techniques ([Ha92]) take time

$$T_{MC}(n, c, m, l, N) = O((n + cm)lN),$$

if there are, on average, N samples, involving random walks of average length l, to determine the mc components in a subset of c rows of X. The last result is obtained for dense matrices and does not take into account the sparsity of the matrix.

In case **(b)** we suppose A is diagonalizable, $X^{-1}AX = diag(\lambda_1, \dots, \lambda_n)$, where $X = (x_1, \dots, x_n)$, and $|\lambda_1| > |\lambda_2| \geq \dots \geq |\lambda_{n-1}| > |\lambda_n|$. The well known *power method* ([GV83]) gives an estimate for the dominant eigenvalue λ_1, which is called the *Raleigh quotient*:

$$\lambda_{max} = \frac{x_k^T x_{k+1}}{x_k^T x_k},$$

where $x_{k+1} = Ax_k$. For the case we want to compute the smallest eigenvalue, the *power method* is altered in the following way: the iteration $x^{new} = Ax^{old}$ is replaced by $x^{new} = Bx^{old}$, where A and B have the same eigenvectors, but different eigenvalues. Letting σ denote a scalar, there are three common choices for B: $B = A - \sigma I$ 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.* When implementing the inverse power method, the inverse power iteration $x^{new} = A^{-1}x^{old}$ is expressed in the form $Ax^{new} = x^{old}$. Replacing A by its LU factorization yields

$$(LU)x^{new} = x^{old}. \qquad (5)$$

In each iteration of the inverse power method, the new x is obtained from the old x by forward and back solving the factored system (5). In a similar manner, the inverse shifted power iteration $x^{new} = (A - \sigma I)^{-1}x^{old}$ is expressed in the form $(A - \sigma I)x^{new} = x^{old}$. If the shifted matrix $A - \sigma I$ is LU factored, then the inverse shifted power iteration assures the form (5).

2 Background - the Monte Carlo method for linear algebra problems

The basic idea of Monte Carlo methods consists in the following: for the problem under consideration a random process is built with the property that the random variables created give the approximate solution of the problem. Generally speaking, the random process is not unique.

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 [So73] 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 $r_N \approx 0.6745 D(\theta_N) N^{-1/2}$.

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

Consider the Markov chain $k_0 \to k_1 \to \ldots \to k_i$, where $k_j = 1, 2, \ldots, n$ for $j = 1, \ldots, i$ are natural numbers. The rules for constructing the chain are:

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

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

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:

$$W_0 = \frac{h_{k_0}}{p_{k_0}}, \ W_j = W_{j-1}\frac{a_{k_{j-1}k_j}}{p_{k_{j-1}k_j}}, \ j = 1, \ldots, i. \tag{6}$$

It is shown [So73], that under the conditions for convergence **(i)**, the following equalities are fulfilled:

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

$$E\{\sum_{i=0}^{\infty} W_i f_{k_i}\} = (h, u), \ \ E\{\sum_{i|k_i=r'} W_i\} = c_{rr'},$$

where $(i|k_i = r')$ means a summation only for weights W_i for which $k_i = r'$ and $C = \{c_{rr'}\}_{r,r'=1}^{n}$is the inverse of A;

$$\frac{E\{W_i f_{k_i}\}}{E\{W_{i-1} f_{k_{i-1}}\}} \approx \lambda_1(A), \quad \text{for sufficiently large "i"}.$$

Instead of simple iterations $Af \in I\!R^{n\times 1}$ we consider *resolvent iterations* $R_q f \in I\!R^{n\times 1}$, where the so-called *resolvent matrix* $R_q(A,q)$ depends on the matrix A and some parameter q. The parameter q will be used to accelerate the convergence of the MC iterations.

There is no universal strategy to define such a resolvent matrix $R_q(A,q)$ in order to accelerate the convergence of all possible linear algebra MC algorithms. In order to do it in an efficient way we need some information about the spectrum of the matrix A (in some cases the needed information is known as a priori information, or can be easily obtained). It is important to note that the resolvent matrix R_q as well as R_q^m (m is any natural number) can be presented as infinite series

$$R_q = \sum_{i=1}^{\infty} c_i A^i, \;\; R_q^m = \sum_{i=1}^{\infty} k_i A^i. \tag{7}$$

The existence of such representations allows to apply efficient MC algorithms since the MC technique has a high efficiency when linear functionals of the powers of the matrices are considered. In fact, when we apply the resolvent MC algorithms we truncate the sequences (7) after η terms, so that we iterate with some approximation of R_q and R_q^m. Practically, the resolvent MC algorithms look similar to the simple MC algorithm. The only difference is that in the resolvent algorithms we use coefficients c_i and k_i obtained in representations (7). Every algorithm based on the use of the resolvent matrix we call *resolvent MC (RMC) algorithm.* In this work we continue the analysis of RMC algorithms started in [DK96] and develop a common MC numerical approach for solving linear algebra problems based on the use of resolvent matrices.

3 Resolvent Monte Carlo algorithms for SLAE

In this section we consider Monte Carlo algorithms for solving linear systems of equations and matrix inversion in the case when the corresponding Neumann series does not converge, or converges slowly.

To analyze the convergence of MC algorithms let us rewrite the equation (3) in the following form:

$$u - qLu = f, \tag{8}$$

where $L = \{l_{ij}\}_{i,j=1}^{n} \in I\!R^{n\times n}$, $f = \{f_i\}_{i=1}^{n} \in I\!R^{n\times n}$ and q is some parameter. Define the resolvent matrix R_q by the equation $I + qR_\lambda = (I - qL)^{-1}$, where I is the *identity matrix*. Let $\lambda_1, \lambda_2, \ldots$ be the eigenvalues of the equation (8), where it is supposed that $|\lambda_1| \geq |\lambda_2| \geq \ldots$. Monte Carlo algorithms are based on the representation $u = (I - qL)^{-1}f = f + qR_q f$, where

$$R_q = L + qL^2 + \ldots\ , \tag{9}$$

The systematic error of (9) when η terms are used is

$$r_s = O[(|q|/|\lambda_1|)^{\eta+1}\eta^{\rho-1}], \tag{10}$$

where ρ is the multiplicity of the roots of λ_1.

¿From (10) is follows that when q is approximately equal to λ_1 the sequence (9) and the corresponding Monte Carlo algorithm converges slowly. When $q \geq \lambda_1$ the algorithm does not converge.

Obviously, the representation (9) can be used for $q : |q| < |\lambda_1|$ to achieve convergence.

Using the resolvent approach we can show how to accelerate the convergence of the MC algorithm. Suppose the series (9) converges slowly or does not converge. We apply a *mapping* of the spectral parameter q in (8). Consider the problem of constructing the solution of (8) for $q \in \Omega$ and $q \neq \lambda_k, k = 1, 2, \ldots$, where the domain Ω is a domain lying inside the definition domain of the $R_q f$, such that all eigenvalues are outside of the domain Ω. In the neighborhood of the point $q = 0$ ($q = 0 \in \Omega$) the resolvent can be expressed by the series $R_q f = \sum_{k=0}^{\infty} c_k q^k$, where $c_k = L^{k+1}f$ is obtained using the simple iterations.

Consider the variable α in the unit circle on the complex plane $\Delta(|\alpha| < 1)$. The function $q = \psi(\alpha) = a_1\alpha + a_2\alpha^2 + \ldots$, maps the domain Δ into Ω. Now it is possible to use the following resolvent

$$R_{\psi(\alpha)}f = \sum_{k=0}^{\infty} b_k \alpha^k\ , \tag{11}$$

where $b_j = \sum_{k=1}^{j} d_k^{(j)} c_k \quad \textit{and} \quad d_k^{(j)} = \frac{1}{j!}\left[\frac{\partial^j}{\partial\alpha^j}[\psi(\alpha)]^k\right]_{\alpha=0}$.

It is clear, that the domain Ω can be chosen so that it will be possible to map the value $q = q_*$ into point $\alpha = \alpha_* = \psi^{-1}(q_*)$ for which the sequence (11) converges; hence the solution of the functional equation (8) can be presented in the form $u = f + q_* R_{\psi(\alpha_*)} f$, where the corresponding sequence for $R_{\psi(\alpha)} f$ converges absolutely and uniformly in the domain Δ.

To apply this approach one needs some information about the spectrum of the matrix. Let us assume, for example, that all eigenvalues λ_k are real and $\lambda_k \in (-\infty, -a]$, where $a > 0$. (The case $\lambda_k \in (-a, a)$ is easy to handle - we use MC method with iteration matrix $1/aL$.) Consider a mapping for the case of interest $(q = q_* = 1)$:

$$\lambda = \psi(\alpha) = \frac{4a\alpha}{(1-\alpha)^2}. \tag{12}$$

The sequence $R_{\psi(\alpha)} f$ for the mapping (12) converges absolutely and uniformly [KA77]. In Monte Carlo calculations we truncate the sequence in (11) after η terms

$$R_{q_*} f \approx \sum_{k=1}^{\eta} b_k \alpha_k^k = \sum_{k=1}^{\eta} \alpha_*^k \sum_{i=1}^{k} d_i^{(k)} c_i = \sum_{k=1}^{\eta} g_k^{(\eta)} c_k,$$

where

$$g_k^{(\eta)} = \sum_{j=k}^{\eta} d_k^{(j)} \alpha_*^j. \tag{13}$$

The coefficients $d_k^{(j)} = (4a)^k C_{k+j-1}^{2k-1}$ and $g_k^{(\eta)}$ can be calculated in advance. In order to calculate the iterations $c_k = L^{k+1} f$ a Monte Carlo algorithm has to be used.

Now consider the problem of evaluating the inner product (1) $J(u) = (h, u) = \sum_{\alpha=1}^{n} h_\alpha u_\alpha$ of a given vector h with the vector solution of the system (2). Define the random variable $\theta_\eta^*[h]$

$$\theta_\eta^*[h] = \frac{h_{k_0}}{p_0} \sum_{\nu=0}^{\eta} g_\nu^{(\eta)} W_\nu f_{k_\nu}, \tag{14}$$

where $W_0 = 1$, $g_0^{(\eta)} = 1$ and $W_\nu = W_{\nu-1} \frac{l_{k_{\nu-1},k_\nu}}{p_{k_{\nu-1},k_\nu}}, \nu = 1, 2, \ldots,$ $(k_0, k_1, k_2, \ldots$ is a Markov chain with initial density function p_{k_0} and transition density function $p_{k_{\nu-1},k_\nu})$ and coefficients $g_j^{(\eta)}$ are defined by (13) for $j \geq 1$. Then

$$E\left\{\lim_{\eta\to\infty} \frac{h_{k_0}}{p_0} \sum_{\nu=0}^{\eta} g_\nu^{(\eta)} W_\nu f_{k_\nu}\right\} = (h, u)$$

and the corresponding Monte Carlo algorithm is given by

$$u_r \approx \frac{1}{N}\sum_{j=1}^{N} \theta_\eta^*[h]_j,$$

where N is the number of chains and $\theta_\eta^*[h]_j$ is the j-th value of $\theta_\eta^*[h]$ defined by (14).

The same approach is used to calculate the inverse matrix. To find the inverse $C = \{c_{rr'}\}_{r,r'=1}^n$ of some matrix A we must first compute the elements of the matrix $L = I - A$, where I is the identity matrix. Clearly the inverse matrix is given by $C = \sum_{i=0}^{\infty} L^i$, which converges if $\|L\| < 1$. If the last condition is not fulfilled or if the corresponding Neumann series converges slowly we can use the same technique for accelerating the convergence of the algorithm. Estimate the element $c_{rr'}$ of the inverse matrix C Let the vector f given by (8) be the following unit vector $f_{r'} = e(r') = (0,\dots,0,1,0,\dots,0)^t$ (where one is in the r' position). Then

$$E\left\{\lim_{\eta\to\infty}\sum_{\nu=0}^{\eta} g_\nu^{(\eta)} \frac{l_{rk_1}l_{k_1k_2}\dots l_{k_{\nu-1}k_\nu}}{p_{rk_1}p_{k_1k_2}\dots p_{k_{\nu-1}p_\nu}} f_{r'}\right\} = c_{rr'}.$$

The last result permits the use of the following Monte Carlo algorithm for calculating elements of the inverse matrix C:

$$c_{rr'} \approx \frac{1}{N}\sum_{j=1}^{N}\left[\sum_{(\nu|k_\nu=r')}^{\eta} g_\nu^{(\eta)} \frac{l_{rk_1}l_{k_1k_2}\dots l_{k_{\nu-1}k_\nu}}{p_{rk_1}p_{k_1k_2}\dots p_{k_{\nu-1}p_\nu}}\right]_j,$$

where $(\nu|k_\nu = r')$ means that only the variables $W_\nu^{(\eta)} = g_\nu^{(\eta)} \frac{l_{rk_1}l_{k_1k_2}\dots l_{k_{\nu-1}k_\nu}}{p_{rk_1}p_{k_1k_2}\dots p_{k_{\nu-1}p_\nu}}$ for which $k_\nu = r'$ are included in the sum.

Observe that since $W_\nu^{(\eta)}$ is only contained in 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, an important saving in computation which we make use of later.

4 Monte Carlo algorithms for computing eigenvalues based on resolvent matrix iterations

Here an algorithm for computing eigenvalues based on Monte Carlo iterations of the matrix A resolvent operator $R_q = [I - qA]^{-1}$ is presented (it is clear that the matrices A and R_q are linear operators). The following representation

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

is valid because of the behaviors of binomial expansion and the spectral theory of linear operators [KA77]. The eigenvalues of the matrices R_q and A are connected with the equality $\mu = \frac{1}{(1-q\lambda)}$, and the eigenfunctions coincide. The following expression

$$\mu^{(m)} = \frac{(R_q^m f, h)}{(R_q^{(m-1)} f, h)} \underset{m\to\infty}{\longrightarrow} \mu = \frac{1}{1-q\lambda}, \quad f \in I\!R^{n\times 1}, h \in I\!R^{n\times 1}.$$

is valid. For a negative value of q, the largest eigenvalue $\mu_{max} = \mu_1$ of R_q corresponds to the smallest eigenvalue $\lambda_{min} = \lambda_n$ of the matrix A. The following statement is valid:

Let λ'_{max} be the largest eigenvalue of the matrix $A' = \{|a_{ij}|\}_{i,j=1}^n$ If q is chosen such that $|\lambda'_{max} q| < 1$, then

$$(R_q^m f, h) = E\{\sum_{i=0}^{\infty} q^i C_{m+i-1}^i W_i h(x_i)\}. \tag{15}$$

where $W_0 = \frac{h_{k_0}}{p_{k_0}}$ and W_i are defined by (6).

The above statement permits the formulation of the Resolvent MC (RMC) algorithm for computing eigenvalues. After some calculations one can obtain

$$\lambda \approx \frac{1}{q}\left(1 - \frac{1}{\mu^{(m)}}\right) = \frac{(AR_q^m f, h)}{(R_q^m f, h)} = \frac{E\{\sum_{i=1}^{\infty} q^{i-1} C_{i+m-2}^{i-1} W_i h(x_i)\}}{E\{\sum_{i=0}^{\infty} q^i C_{i+m-1}^i W_i h(x_i)\}}.$$

The coefficients C_{n+m}^n are calculated using the formula: $C_{i+m}^i = C_{i+m-1}^i + C_{i+m-1}^{i-1}$. ¿From the representation $\mu^{(m)} = \frac{1}{1-|q|\lambda^{(m)}} \approx \frac{(h, R_q^m f)}{(h, R_q^{(m-1)} f)}$ we obtain the following RMC algorithm for evaluating the smallest eigenvalue:

$$\lambda \approx \frac{1}{q}\left(1 - \frac{1}{\mu^{(m)}}\right) \approx \frac{E\{\sum_{i=0}^{l} q^i C_{i+m-1}^i W_{i+1} h(x_i)\}}{E\{\sum_{i=0}^{l} q^i C_{i+m-1}^i W_i h(x_i)\}},$$

where $W_0 = \frac{h_{k_0}}{p_{k_0}}$ and W_i are defined by (6).

Since the initial vector f can be any vector $f \in I\!R^{n\times 1}$ (in particular, a unit vector), the following formula for calculating λ_{min} is used

$$\lambda \approx \frac{E\{W_1 + qC_m^1 W_2 + q^2 C_{m+1}^2 W_3 + \ldots + q^l C_{l+m-1}^l W_{l+1}\}}{E\{1 + qC_m^1 W_1 + q^2 C_{m+1}^2 W_2 + \ldots + q^l C_{l+m-1}^l W_l\}},$$

that is

$$\lambda \approx \frac{\frac{1}{N}\sum_{j=1}^{N}\{\sum_{i=0}^{l} q^i C_{i+m-1}^{i} W_{i+1}\}_j}{\frac{1}{N}\sum_{j=1}^{N}\{\sum_{i=0}^{l} q^i C_{i+m-1}^{i} W_{i}\}_j}.$$

Using the same presentations for a positive value of q we formulate the RMC algorithm for calculating the dominant eigenvalue.

5 Parallel implementation

In this section a parallelization of the Monte Carlo algorithms is considered. Estimates for time (algorithm complexity), speedup and parallel efficiency are obtained. These estimates are confirmed by numerical tests performed on a cluster of workstations. We establish the results illustrating:

- high parallel efficiency and good speedup;
- independence between computing time and matrix size for large sparse matrices.

Advantages and disadvantages of the MC algorithms in the context of its parallel realization using MPI are presented and discussed.

5.1 Time, speedup and parallel efficiency estimations

Consider a multiprocessor configuration consisting of p nodes (processors). Every node performs its own instructions on the data in its own memory. The inherent parallelism of the Monte Carlo methods lies in the possibility of calculating each realization of the random variable θ on a different processor. There is no need for communication between the nodes during the time of calculating the realizations - the only need for communication occurs at the end when the averaged value is to be calculated.

We consider the *parallel efficiency* Ef as a measure characterizing the quality of the proposed algorithms. We use the following definition:

$$Ef_p(X) = \frac{S_p(X)}{p} = \frac{ET_1(X)}{pET_p(X)},$$

where by X we denote the Monte Carlo algorithm, $ET_i(X)$ is the expected value of the computational time for implementing the algorithm X on a system of p nodes and $S_p(X)$ is the speedup of the algorithm X realized on p processors.

Every move in a Markov chain is done according to the following algorithm:

- **(i)** generation of a random number (it is usually done in k arithmetic operations where $k = 2$ or 3);
- **(ii)** determination of the next element of the matrix : this step includes a random number of logical operations [2];
- **(iii)** calculating the corresponding random variable.

Since the Monte Carlo Almost Optimal (MAO) algorithm is used (see, [Di86]), the random process never visits the zero-elements of the matrix A. (This is one of the reasons why MAO algorithm has high algorithmic efficiency for sparse matrices.) Let d_i be the number of non-zero elements of the i-th row of the matrix A. Obviously, the number of logical operations γ_L in every move of the Markov chain can be estimated using the following expression

$$E\gamma_L \approx \frac{1}{2}\frac{1}{n}\sum_{i=1}^{n} d_i = \frac{1}{2}d. \tag{16}$$

Since γ_L is a random variable we need an estimation of the probable error of (16). It depends on the balance of the matrix. For matrices which are not very disbalanced and of not very small-size, the probable error of (16) is negligible small in comparison with γ_L. The number of arithmetic operations, excluding the number of arithmetic operations k for generating the random number is γ_A. The mathematical expectation of the operations needed for each move of any Markov chain is

$$E\delta = (k + \gamma_A)s_A + \frac{1}{2}ds_L,$$

where s_A and s_L are the numbers of sub-operations of the arithmetic and logical operations, respectively. In order to obtain the initial density vector and the transition density matrix, the algorithm needs d_i multiplications for obtaining the i-th row of the transition density matrix and $2dn$ arithmetic operations for constructing the transition density matrix $\{p_{\alpha\beta}\}_{\alpha,\beta=1}^{n}$, where d is determined by (16). Thus, the mathematical expectation of the computational complexity (total time of the algorithm) becomes

$$ET_1(RMC) \approx 2\left[(k + \gamma_A)s_A + \frac{1}{2}ds_L\right] lN + 2n(1 + d)s_A, \tag{17}$$

[2] Here the logical operation deals with testing the inequality "$a < b$".

where l is the numbers of moves in every realization of the Markov chain, and N is the number of realizations. It is worth noting that the main term of (17) does not depend on the size n of the matrix. It depends linearly on the mean value of the number of the non-zero elements per row. This result means that the time required for calculating the eigenvalue of a sparse matrix by RMC practically does not depend on n. The parameters l and N depend on the spectrum of the matrix, but do not depend on the size n. The above mentioned result was confirmed for a wide range of matrices during the actual numerical experiments. It is easy to see that the main term in our estimate (17) coincides with Halton's result [Ha92] in the case of dense matrices, because of $d = n$.

Now one can estimate the speedup $S_p(RMC)$ of a multiprocessor with p nodes

$$S_p(RMC) \approx \frac{\left[(k+\gamma_A)s_A + \frac{1}{2}ds_L\right] lN + n(1+d)s_A}{\left[(k+\gamma_A)s_A + \frac{1}{2}ds_L\right] l\frac{N}{p} + n\left[1+\frac{d}{p}\right] s_A}.$$

Suppose that $\left[(k+\gamma_A)s_A + \frac{1}{2}ds_L\right] lN = \frac{1}{\varepsilon}n(p+d)s_A$, where ε is a small positive number. Then for every $p \geq 1$

$$S_p(RMC) \geq \frac{p}{1+\varepsilon} \geq 1.$$

For the parallel efficiency we have: $\frac{1}{1+\varepsilon} \leq Ef_p(RMC) \leq 1$. The last inequality shows that the parallel efficiency of RMC algorithm can be really very close to 1.

5.3 Numerical tests

The numerical tests are made on a cluster of 48 Hewlett Packard 900 series 700 Unix workstations under MPI (version 1.1), [MPI]. 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 nodes). 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 stored in *packed row format* (i.e. only nonzero elements). The results for the average time of the algorithm to compute the eigenvalues are given in Tables 1 and 2. The relative accuracy of all calculated

Table 1
Implementation of **simple iterative MC algorithm** using MPI (number of trajectories - $N = 10^5$).

Number of nodes	1	2	3	4	5
Time	$T(ms)$	$T(ms)$	$T(ms)$	$T(ms)$	$T(ms)$
matrix n = 128	34	17	11	8	7
matrix n = 1024	111	56	37	27	21
matrix n = 2000	167	83	56	42	35

Table 2
Implementation of **Resolvent Monte Carlo Algorithm** for evaluation of λ_{max} using MPI (number of trajectories - $N = 10^5$; $q > 0$).

Number of nodes	1	2	3	4	5
Time	$T(ms)$	$T(ms)$	$T(ms)$	$T(ms)$	$T(ms)$
matrix n = 128	18	9	6	4	3
matrix n = 1024	30	15	10	7	6
matrix n = 2000	21	11	7	5	4

values is 10^{-3}. The results show that the speedup is almost linear. For some calculations we have a superlinear speedup which could be explained by the more efficient use of the high RAM memory when the number of processor increases.

Our numerical tests of solving linear systems as well as for computing eigenvalues of real symmetric sparse matrices show that the computational time of the resolvent MC algorithms decreases with a factor of 2 to 8 times in comparison with the time of the corresponding simple iterative MC. The factor of improvement of the complexity depends on the mean value of the number of non-zero elements per row of the original and resolvent matrices as well as of the norms of the resolvent and original matrices and does not depend on the size of the matrices. This factor also depends on the truncation error of the series presenting the resolvent matrix using η terms. We do this comparison for the same accuracy reached by using both approaches - the simple iterative MC and the resolvent MC.

6 Concluding remarks

For solving linear algebra problems we consider a common approach based on resolvent matrix MC iterations. This approach can be applied if some a priori information is available. It improves the convergence of the algorithms. All we need is to find the coefficients in the representation of the resolvent matrix and use them to define the random variable used in the calculations.

We analyze the computational complexity, speedup and efficiency of the resolvent algorithm in the case of dealing with sparse matrices. It is shown that for sparse matrices the computational complexity depends linearly on the mean value of the number of non-zero elements per row and does not depend on the size of the matrix. As a special case our result coincides with Halton's result obtained for dense matrices.

Our numerical results obtained on a cluster of computers using MPI for large sparse matrices demonstrate linear speedup and high parallel efficiency of the algorithms.

References

[Di86] Dimov I.T., *Minimization of the probable error for Monte Carlo methods.* **Application of Mathematics in Technology. Differential equations and applications.** Varna 1986, Sofia 1987, pp. 161–164.

[DK98] I. Dimov, A. Karaivanova *Parallel computations of eigenvalues based on a Monte Carlo approach*, **Journal of MC Methods and Appl.**, Vol.4,Num.1, 1998 pp.33-52.

[DK96] I. Dimov and A. Karaivanova, *Iterative Monte Carlo algorithms for linear algebra problems*, **Lecture Notes in Computer Science**, 1196 (1996), Springer, 1996, pp.66-77.

[GV83] G. H. Golub, C.F. Van Loon, *Matrix computations*, **The Johns Hopkins Univ. Press**, Baltimore, 1983.

[Ha92] 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.

[KA77] L.V. Kantorovich, G.P. Akilov, *Functional analysis*, **Nauka**, Moscow, 1977.

[MAD94] G.Megson, V.Aleksandrov, I.Dimov, *Systolic Matrix Inversion Using a Monte Carlo Method*, **Journal of Parallel Algorithms and Applications, vol.4**, No 1 (1994).

[MPI] Message Passing Interface Forum, *MPI: A Message- passing Interface Standard*, **Technical report CS-94-230**, Computer Science Dept., University of Tennessee, Knoxville, TN, 1994.

[Mi70] 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.

[Mi87] G.A. Mikhailov, *Optimization of the "weight" Monte Carlo methods*, **Nauka**, Moscow, 1987.

[So73] I.M. Sobol *Monte Carlo numerical methods*, **Nauka**, Moscow, 1973.

[1] C. Addison, B.Beattie, N. Brown, R. Cook, B. Stevens and D. Watson, *Distributed objects: sequential objects with parallel performance* In Proc. 6th SIAM Conf Parallel Processing fof Sci. Comput., 1993.

[2] Megson G.M. , Aleksandrov V.N. and Dimov I.T. Systolic Matrix Inversion Using a Monte Carlo Method, J. Parallel Algorithms and Applications, Vol.3, No3

[3] Dimov I.T., Megson G.M. , Aleksandrov V.N. A New Highly Convergent Monte Carlo Method for Matrix Computations and its Implementation on Fast Area-efficient Systolic Array Technical Report, Dept. Computing Science, University of Newcastle, 1995.

[4] Bertsekas D.P. and Tsitsiklis , *Parallel and Distributed Computation* , Prentice Hall, 1989

[5] T. Hey and J. Ferrante, *Portability and Performance for Parallel Processing*, John Willey and Sons, Chichester, New York, 1994

[6] Westlake J.R., *A Handbook of Numerical Matrix Inversion and Solution of Linear Equations*, John Wiley and Sons, New York, 1968.

[7] S. G. Akl, *The design and analysis of parallel algorithms*, Prentice-Hall, Inc., 1989.

[8] Dimov I.T. Minimization of the probable error for some Monte Carlo methods - Proc. of the Summer School on Mathematical Modelling and Scientific Computations, 23-28.09.1990, Albena, Bulgaria, Publ. House of the Bulg. Acad. Sci., 1991, pp. 159- 170.

[9] I. Dimov , Efficient and Overconvergent Monte Carlo Methods. Addvances in Parallel Algorithms, IOS Press, Amsterdam, 1994, pp. 100-111.

[10] Dimov I., Tonev O. Ranfom Walk on Distant Mesh Points Monte Carlo Methods, Journal of Statistical Physics, vol. 70, Nos.5/6, 1993, pp. 1333-1342.

[11] Dimov I.T., Tonev O.I. Monte Caralo algorithms: performance analysis for some computer architectures, Journal of Computational and Applied Methematics, vol. 48 (1993), pp.253-277.

[12] Dimov I.T., Tonev O.I. Monte Carlo numerical methods with overconvergent probable error. Numerical Methods and Applications, Proc. of the Intern. Conf. on Numerical Methods and Apllications, Sofia, Publ. House of the Bulg. Acad. Sci., Sofia, 1989, pp.116-120.

[13] I.Dimov Criteria estimators for Speed-up and Efficiency of some parallel Monte Carlo algorithms for different computer architectures. Proc. WP&DP91 , North-Holland, Amsterdam, 1992, pp.243-262.

[14] I. Dimov, A. Karaivanova, Monte Carlo Parallel Algorithms.Proc. of the Third Intern. Conference on Applications of Supercomputers in Engineering ASE/93, Comput. Mech. Publ., Elsevier Applied Sci., London, New York, 1993, pp. 479-495.

[15] Z.Zlatev, I.Dimov, K.Georgiev. Modeling the Long-Range Transport of Air Pollutants, IEEE Computational science and engineering, No 5 (1994), pp. 45-52.

[16] S.M. Ermakov, V.V. Nekrutkin and A.S. Sipin, *Random processes for solving classical equations of the mathematical physics*, Nauka, Moskow, 1984 (in Russian).

[17] N. J. Sørensen and B. S. Andersen, *A parallel finite element method for the analysis of crystalline solids*, Computer Methods in Applied Mechanics and Engineering; to appear.

[18] M. Hanke & P. C. Hansen, *Regularization methods for large-scale problems*, Surv. Math. Ind. **3** (1993), 253–315.

[19] P. C. Hansen, *Experience with regularizing CG iterations;* in M. Natori & T. Nodera (Eds.), *Matrix Analysis and Parallel Computing*, Keio University, 1994; pp. 48–59.

[20] M. Hanke, *Conjugate Gradient Type Methods for Ill-Posed Problems*, Pitman Research Notes in Mathematics, Harlow, 1995.

[21] Z. Zlatev, P. C. Hansen, Tz. Ostromsky and A. Sameh, *Using large dense blocks in sparse QR factorizations*, submitted to BIT.

[22] Å. Björck, E. Grimme & P. M. Van Dooren, *An implicit shift bidiagonalization algorithm for ill-posed systems*, BIT **34** (1994), 510–534.

[23] C. C. Paige & M. A. Saunders, *LSQR: an algorithm for sparse linear equations and sparse least squares*, ACM Trans. Math. Software **8** (1982), 43–71.

[24] A. Quarteroni, *Domain decomposition and parallel processing for the numerical solution of partial differential equations*, Surveys on Mathematics for Industry **1** (1991), 75–118.

[25] T.L. Casavant and J.G. Kuhl. A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems In *IEEE Transactions on Software Engineering*, Vol. 14, No. 2, February 1988.

[26] Yves Arrouye. Scope: An Extensible Interactive Environment for the Performance Evaluation of Parallel Systems. *Microprocessing and Microprogramming Journal*, December 1995. Hors-série *Parallel Systems Engineering.*

[27] E. Bampis, Frédéric Guinand, and D. Trystram. Some models for scheduling parallel programs with communication delays. *Discrete Applied Mathematics*, (to appear), 1996.

[28] E. Bampis, J.-C. Konig, and D. Trystram. Impact of communications on the complexity of the parallel gaussian elimination. *Parallel Computing*, 13, 1991.

[29] Michel Christaller, Jacques Briat, and Michel Rivière. ATHAPASCAN-0: concepts structurants simples pour une programmation parallèle efficace. *Calculateurs Parallèles*, 7(2):173–196, 1995.

[30] L. Colombet, P. Michallon, and D. Trystram. Parallel matrix-vector product on rings with a minimum of communications. *Parallel Computing*, (to appear), 1996.

[31] J. Chassin de Kergommeaux. Mise au point d'applications parallèles à partir de traces d'exécution. *La lettre des calculateurs distribués - numéro spécial Actes de l'Ecole SPI Lyon juillet 94*, 1994.

[32] Frédéric Guinand. *Ordonnancement avec communications dans les environnements de programmation parallèles.* Thèse, Institut National Polytechnique de Grenoble, France, juin 1994.

[33] J. Chassin de Kergommeaux, editor. *Environnements d'Exécution de programmes parallèles.* HERMS, 1995. Numéro spécial de *Calculateurs Parallèles*, Vol. 7, No.

[34] D. Delesalle, L. Desbat, and D. Trystram. Résolution de grands systèmes creux par méthodes itératives parallèles. *Mathematical Modelling and Numerical Analysis*, 27(6):651–671 1 993.

[35] B. Dumitrescu, J.-L. Roch, and D. Trystram. Fast matrix multiplications algorithms on mimd architectures. *Parallel Algorithms and Applications*, 4(2), 1994.

[36] J.-L. Roch and D. Trystram. Parallel Winograd Multiplication. In W. Joosen and E. Milgrom, editors, *EWPC'92, Parallel Computing: from theory to sound practice*, pages 578–581, Barcelona, 1992. IOS Press.

[37] J.-L. Roch, A. Vermeerbergen, and G. Villard. A new load-prediction scheme based on algorithmic cost functions. In *CONPAR 94, Linz Austria*, LNCS 854, 1994.

[38] J.-L. Roch, A. Vermerbergen, and G. Villard. Cost prediction for Load Balancing. In *Proceedings of CONPAR'92*, Lyon, Septembre 1992. Springer-Verlag.

[39] Jacques Briat, Jacques Chassin de Kergommeaux, Brigitte Plateau, Jean-Louis Roch, Denis Trystram, Gilles Villard, and Jean-Marc Vincent. APACHE : Algorithmique Parallèle et pArtage de CHargE In *Actes des* 5^{iemes} *Rencontres sur le Parallélisme - RenPar5*, Brest, Mai 1993.

[40] M. Christaller. ATHAPASCAN-0A control parallelism approach on top of pvm. In *Proc PVM User's group meeting.* University of Tenness BADDDDD

[41] M. Christaller. ATHAPASCAN-0B for pvm: adding threads to pvm. In *Proc. EuroPVM User's Group*, October 1994.

[42] Michel Christaller, Martha-Rosa Castaneda Retiz, and Thierry Gautier. Control parallelism on top of PVM: The ATHAPASCAN environment. In Jack Dongarra, Marc Gengler, Bernard Tourancheau, and Xavier Vigouroux, editors, *Proc. Second European PVM User's Group Meeting*, pages 71–76, Ecole Nationale Supérieure, Lyon, France, September 1995. Hermes.

[43] L. Colombet, L. Desbat, and F. Ménard. Star modeling on IBM RS6000 networks using PVM. In *Proceedings of the 2nd International Symposium on Hig Performance Distributed Computing*, 1993.

[44] A. Fagot and J. Chassin de Kergommeaux. Optimized record-replay for rpc-based parallel programming. In *Working conference on programming environments for massivel p arallel distributed systems*, pages 347–352, Ascona, Switzerland, April 1994. IFIP, WG10.3, Birkhaeuser, Basel.

[45] A. Fagot and J. Chassin de Kergommeaux. Formal and experimental validation of a low-overhead execution repl mechanism. In S. Haridi, K. Ali, and P. Magnusson, editors, *Euro-Par'95 Parallel Processing*, volume 966 of *LNCS*, pages 167–178. Springer-Verlag, August 1995.

[46] A. Fagot and J. Chassin de Kergommeaux. Systematic assessment of the overhead of tracing parallel programs. In E.L. Zapata, editor, *Proceedings of the 4th Euromicro Workshop on Parallel and Distributed processing, PDP'96*, Braga, January 1996. IEEE/CS.

[47] Y. Trémolet, F-X. Le Dimet, and D. Trystram. Parallelization of scientific applications: Data assimilation in meteorology. In W. Gentzsch, editor, *Proceedings of HPCN'94*, number 796 in LNCS, Muenchen, April 1994. Springer Verlag.