Parallel computations of eigenvalues based on a Monte Carlo approach

Ivan Dimov and Aneta Karaivanova

Central Laboratory for Parallel Computing
Bulgarian Academy of Sciences
Acad. G. Bonchev St.,bl. 25 A, Sofia, 1113, Bulgaria
E-mail: anet@amigo.acad.bg, dimov@amigo.acad.bg
Web site: http://www.acad.bg/BulRTD/math/dimov2.html

Key words: Monte Carlo algorithms, eigenvalue, efficiency estimator, Markov chain, parallel algorithm

Abstract

This paper presents a parallel Monte Carlo algorithm for evaluating the eigenvalues of matrices. This algorithm is called Resolvent Monte Carlo algorithm (RMC) and uses Monte Carlo iterations by the resolvent matrix. The algorithm is suitable for estimating eigenvalues of large sparse symmetric matrices.

The works described in this paper have the goal to reduce the computational time when parallel machine is used and to reach good experimental results for speedup and parallel efficiency for large sparse matrices.

Numerical tests are performed for a number of test matrices - general sparse symmetric, band symmetric matrices and dense matrices on the parallel machine Intel PARAGON. Estimations for the speedup as well as for the parallel efficiency are obtained. It is shown that the algorithm complexity is practically independent of the size of the matrix.

1 Introduction

Let A be a real symmetric matrix. Consider the problem of evaluating the eigenvalues of A, i.e. the values of λ for which

$$Au = \lambda u \tag{1}$$

holds.

It is known that the problem of calculating the smallest eigenvalue of A is more difficult from numerical point of view than the problem of evaluating the largest eigenvalue. Nevertheless, for many applications in physics and engineering it is important to estimate the smallest one, because it usually defines the most stable state of the system which is described by the considered matrix.

There are also, many problems in which it is important to have an efficient parallel algorithm. For large sparse matrices it is not easy to find efficient algorithm for evaluating the smallest eigenvalue. The problem of plotting the spectral portraits of matrices is one of the important problems where high-efficient parallel algorithms are needed. The spectral portraits are used in stability analysis. The above mentioned problem leads to a large number of subproblems of evaluating the smallest eigenvalue of symmetric matrices.

It is well known that Monte Carlo methods are very efficient when parallel computers or processors are available, because they are inherently parallel and have loose 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 traditional numerical methods.

The basic idea of Monte Carlo methods consists in the following: for a given problem a random process is constructed with the property to give an approximate solution under appropriate statistical process. Generally speaking, such random process can be built not in a unique way.

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.

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 (which does not use any additional a priori information about the regularity of the solution) [So73] is defined as parameter r_N for which:

$$Pr\{|J - \theta_N| \geq r_N\} = 1/2 = Pr\{|J - \theta_N| \leq r_N\}. \tag{2}$$

If the standard deviation is bounded, i.e. $\sigma(\theta_N) < \infty$, the normal convergence in the central limit theorem holds.

$$Pr\{|J - \theta_N| \leq x\sigma(\theta_N)N^{-1/2}\} \approx \Phi(x).$$

Obviously, from (2) and $\Phi(0.6745) \approx \frac{1}{2}$ we have

$$r_N \approx 0.6745\sigma(\theta_N)N^{-1/2}. \tag{3}$$

Let $\hat{\theta}_N$ be another random variable for which $J = E\hat{\theta}$ and the conditions providing the existence and the finiteness of the variance $D\hat{\theta_N}$ are satisfied.

A method for which $D\hat{\theta}_N < D\theta_N$ is called *efficient Monte Carlo method* [Ka50], [Mi87], [Di91], [DT93]. A method of this type for evaluation of integrals is proposed by Kahn [Ka50]. This approach is generalized for evaluation of integral equations

[Mi87], [Di91]. The method [Di91], [DT93] with the smallest $D\hat{\theta}_{MO}$ is called *Monte Carlo Optimal Method* (MO). It is shown in [Di91], [MAD94] that methods of this type are time-consuming. From an algorithmic point of view it is more efficient to use *Monte Carlo Almost Optimal method* (MAO) with a variance $D\hat{\theta}_{MAO}$ for which $D\hat{\theta}_{MAO} \geq D\hat{\theta}_{MO}$, but $|D\hat{\theta}_{MAO} - D\hat{\theta}_{MO}| \leq \varepsilon$, where ε is a small parameter and the random variable $\hat{\theta}_{MAO}$ is easy constructed and the simulation of the corresponding random process is less time-consuming (see, for example, [MAD94]).

The algorithm under consideration is an algorithm of the second type, i.e. MAO algorithm. The value of the variance is reduced by special choice of the transition-density matrices.

In this paper a Monte Carlo algorithm is presented with the goal to reduce the computational time when parallel machine is used and to reach good experimental results for speedup and parallel efficiency for large sparse symmetric matrices.

2 The Monte Carlo algorithm

2.1 Background

There are two main approaches in Monte Carlo literature for evaluation the dominant (maximal by modulus) eigenvalue of an integral operator. Let $k(x', x)$ be the kernel of the operator K, where $x', x \in I\!R^n$. It is supposed that $I\!R^n$ is n-dimensional Euclidean space and $K \in [L \to L]$ where L is a Banach functional space. Suppose also that K and K^* have properties providing the convergence of the iteration process:

$$\lambda^{(m)} = (K^m \psi, \varphi)/(K^{(m-1)}\psi, \varphi), \quad \psi \in L, \quad \varphi \in L^* \tag{4}$$

to the dominant eigenvalue $\lambda > 0$. (See, for example, [Vl56])

The operator R_q given by

$$R_q = [I - qK]^{-1}$$

is called *resolvent operator* (see, for example, [Sa89], p. 58).

Let $|\lambda q| < 1$. The eigenvector sets of the operators $[I - qK]^{-1}$ and K coincide and the corresponding eigenvalues $\{\mu_k\}$ and $\{\lambda_k\}$ are connected by the following equation: $\mu_k = (1 - q\lambda_k)^{-1}$ for $q > 0$. Having in mind the same suppositions about the operator K and the functions φ and ψ, when $q > 0$

$$\mu^{(m)} = \frac{([I - qK]^{-m}\psi, \varphi)}{([I - qK]^{-(m-1)}\psi, \varphi)} \to_{m\to\infty} \mu = \frac{1}{1 - q\lambda}. \tag{5}$$

In the Monte Carlo literature there are two algorithms realizing iteration processes (4) ([Vl56], [So73]) and (5) ([Mi70], [Mi87]). In this paper we will present the corresponding algorithms for calculating the smallest eigenvalue of real matrices. First we will define the Markov chains which will be used.

2.2 Markov Chains for Almost Optimal Algorithms for Matrix Computations

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

Consider the following Markov chain:

$$k_0 \to k_1 \to \ldots \to k_i, \tag{6}$$

where $k_j = 1, 2, \ldots, n$ for $j = 1, \ldots, i$ are natural numbers. The rules for constructing the chain (6) are:

$$Pr(k_0 = \alpha) = p_\alpha, \;\; Pr(k_j = \beta | k_{j-1} = \alpha) = p_{\alpha\beta}. \tag{7}$$

From all possible permissible initial and transition densities we choose:

$$p = \{p_\alpha\}_{\alpha=1}^n, \;\; p_\alpha = \frac{|h_\alpha|}{\sum_{\alpha=1}^n |h_\alpha|}; \tag{8}$$

$$P = \{p_{\alpha\beta}\}_{\alpha\beta=1}^n, \;\; p_{\alpha\beta} = \frac{a_{\alpha\beta}}{\sum_{\beta=1}^n |a_{\alpha\beta}|}, \;\; \alpha = 1, \ldots, n. \tag{9}$$

Such a choice of the initial density vector and the transition density matrix leads to MAO algorithms for matrix computations.

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

$$W_0 = \frac{h_{k_0}}{p_{k_0}}, \;\; W_j = W_{j-1} \frac{a_{k_{j-1}k_j}}{p_{k_{j-1}k_j}}, \;\; j = 1, \ldots, i. \tag{10}$$

Following [So73] it is easy to show that

$$E\{W_i f_{k_i}\} = (h, A^i f), \;\; i = 1, 2, \ldots. \tag{11}$$

2.3 Resolvent Monte Carlo Iterations for Evaluating the Smallest Eigenvalue

Given a matrix A, consider an algorithm based on Monte Carlo iterations by the resolvent matrix $R_q = [I - qA]^{-1} \in I\!R^{n \times n}$. The following representation

$$[I - qA]^{-m} = \sum_{i=1}^{\infty} q^i C_{m+i-1}^i, \;\; |q\lambda| < 1 \tag{12}$$

is valid because of behaviors of binomial expansion and the spectral theory of linear operators. Let remaind that the eigenvalues of the matrices R_q and A are connected by the equality $\mu = \frac{1}{1-q\lambda}$, and the eigenvectors coincide. According to (5), the following expression

$$\mu^{(m)} = \frac{([I-qK]^{-m}f,h)}{([I-qK]^{-(m-1)}f,h)} \to_{m\to\infty} \mu = \frac{1}{1-q\lambda}, \quad f \in I\!R^n, h \in I\!R^n, \tag{13}$$

is valid (f and h are vectors in $I\!R^n$). For negative values of q, the largest eigenvalue μ_{max} of R_q corresponds to the smallest eigenvalue λ_{min} of the matrix A. Now, for constructing the method it is sufficient to prove the following theorem.

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

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

(The proof of this theorem follows the proof in [Mi70]. Since the matrices A and $[I-qA]^{-1}$) are linear operators it is easy to show that the needed conditions of the theorem proved in [Mi70] are fulfilled in our case.)

After some calculations one can obtain

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

$$\frac{E\sum_{i=1}^{\infty} q^{i-1}C^{i-1}_{i+m-2}W_i f(x_i)}{E\sum_{i=0}^{\infty} q^i C^i_{i+m-1}W_i f(x_i)} = \frac{E\sum_{i=0}^{l} q^i C^i_{i+m-1}W_{i+1}f(x_i)}{E\sum_{n=0}^{l} q^i C^i_{i+m-1}W_i f(x_i)}, \tag{14}$$

where $W_0 = \frac{h_{k_0}}{p_{k_0}}$ and W_i are defined by (10). The coefficients C^n_{n+m} are calculated using the presentation

$$C^i_{i+m} = C^i_{i+m-1} + C^{i-1}_{i+m-1}.$$

When one is interested in estimating the smallest eigenvalue, the parameter $q < 0$ has to be chosen so that to minimize the following expression

$$J(q,A) = \frac{1+|q|\lambda_1}{1+|q|\lambda_2},$$

or if $\lambda_1 = \alpha\lambda_2, \quad (0 < \alpha < 1)$,

$$J(q,A) = 1 - \frac{|q|\lambda_2(1-\alpha)}{1+|q|\lambda_2}.$$

We choose

$$q = -\frac{1}{2\|A\|}$$

but sometimes a slightly different value of q might give better results when a number of realizations of the algorithm is considered.

This method has strong requirements about matrices for which it can be applied. The systematic error consists of two parts:

- an error from the Power method applied on the resolvent matrix $[I - qA]^{-1}$ - it determines the value of the parameter m in the following way. The rate of convergence is

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

  where μ_1 and μ_2 are the largest two eigenvalues of the resolvent matrix. The parameter m has to be chosen such that $(\frac{\mu_2}{\mu_1})^m < \varepsilon$.

- an error which comes from the series expansion of the resolvent matrix - it determines the value of the parameter l. Considering the presentation

$$([I - qA]^{-m} f, g) \approx \sum_{i=0}^{l} q^i C^i_{m+i-1}(A^i f, g) = \sum_{i=1}^{l} u_i,$$

  the parameter l has to be chosen such that $|u_{i+1}| < \varepsilon_2$.

- rounding errors - to maintain them at a low level the condition $(max_{i=1,l}|u_i|)\varepsilon < \alpha\mu$ must be satisfied, where $\alpha < 1$ represents the requested precision and ε is the machine precision parameter.

Unfortunately, the parameter l can not be relatively large because the binomial coefficients C^l_{m+l-1} grow exponentially with l.

3 Parallel Implementation

In this section a parallelization of the Monte Carlo algorithm is considered. Estimates for time (algorithm complexity), speedup and parallel efficiency are obtained. These estimates are confirmed by numerical tests performed on the parallel computer Intel PARAGON. 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 RMC algorithm in the context of its parallel realization on Intel PARAGON are presented and discussed.

3.1 Time, Speedup and Parallel Efficiency Estimations

Consider a multiprocessor configuration consisting of p processors. Every processor 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 processors during the time of calculating the realizations - the only need for communication occurs at the end when the averaged value is to be calculated.

Let a parallel algorithm X for solving the given problem be given. If X is a deterministic algorithm then $T_p(X)$ is the time (or the number of sub-operations) needed for realizing the algorithm X on a system of p processors. If X is a Monte Carlo algorithm it is impossible to obtain an exact theoretical estimation for $T_p(X)$ since every realization of the algorithm yields different values for $T_p(X)$. Theoretical estimations are possible only for the mathematical expectation, i.e. for $ET_p(X)$. The parameters – *speedup*:

$$S_p(X) = \frac{ET_1(X)}{ET_p(X)}$$

and *parallel efficiency*

$$E_p(X) = S_p(X)/p.$$

(introduced in [DT93a]) characterize the quality of the parallel Monte Carlo algorithm.

Further, we will consider a Monte Carlo algorithm for calculating the smallest and the dominant eigenvalue of symmetric matrices and will get estimations for speedup and parallel efficiency. The algorithm under consideration is an iterative Monte Carlo algorithm. The only difference is that when a negative value of the acceleration parameter q is used the Monte Carlo iterations converge to the smallest eigenvalue and when a positive value of q is used the iterations converge to the dominant eigenvalue. In fact, in both modifications of the algorithm we consider a random process which consists in jumping on element of the matrix.

3.2 Estimations for RMC algorithm

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 [1];

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

- **(iii)** calculating the corresponding random variable.

Since Monte Carlo Almost Optimal (MAO) algorithm is used, 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 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

$$E\gamma_L \approx \frac{1}{2}\frac{1}{n}\sum_{i=1}^{n} d_i = \frac{1}{2}d. \tag{15}$$

Since γ_L is a random variable we need an estimation of the probable error of (15). It depends on the balance of the matrix. For matrices which are not very disbalanced and of not vary small-size ($n = 2, 3$), the probable error of (15) 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

$$E\delta = \tau \left[(k + \gamma_A) l_A + \frac{1}{2} d l_L \right], \tag{16}$$

where l_A and l_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 $\{p_{\alpha\beta}\}_{\alpha,\beta=1}^{n}$, where d is determined by (15).

Thus, the mathematical expectation of the total time of the algorithm becomes

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

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. This result 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 not depend on the size n. The above mentioned result was confirmed for a wide range of matrices during the realized numerical experiments.

For the multiprocessor model with p processors we have

$$S_p(RMC) \approx \frac{\left[(k + \gamma_A) l_A + \frac{1}{2} d l_L \right] lN + n(1 + d)}{\left[(k + \gamma_A) l_A + \frac{1}{2} d l_L \right] l\frac{N}{p} + n\left[1 + \frac{d}{p} \right]}. \tag{18}$$

Suppose that

$$\left[(k+\gamma_A)l_A+\frac{1}{2}dl_L\right]lN=\frac{1}{\varepsilon}n(p+d),$$

where ε is a small positive number. Then for every $p \geq 1$

$$S_p(RMC) \geq \frac{p}{1+\varepsilon} \geq 1. \tag{19}$$

For the parallel efficiency we have

$$\frac{1}{1+\varepsilon} \leq E_p(RMC) \leq 1. \tag{20}$$

The last inequality shows that the parallel efficiency of RMC algorithm can be really very close to 1.

3.3 Numerical Tests

The numerical tests are performed on an Intel PARAGON machine. The Intel PARAGON is a particular form of a parallel machine which consists of a set of independent processors, each with its own memory, capable of operating on its own data. Each processor has its own program to execute and processors are linked by communication channels. The PARAGON machine on which our experiments are performed consists of a mesh-grid of $16 \times 4 = 64$ nodes. 8 processors are devoted to the system management (service and I/O processors) while the 56 remain for applications purposes. An amount of 16 MB of RAM is also available on each node but approximately 8 to 9 are consumed by OSF such that 7 MB are really available for the users. Data needed for processing the PARAGON must be shared between processors by *message passing*. There are a number of libraries available for message passing on the PARAGON system. Here we use the proprietary NX message passing library provided by Intel. This offers the best message passing performance for this system. The Task-to-Task communication latency is about $50\mu s$ while the bandwidth is approximately 92 MB/s under OSF.

In this section numerical tests obtained using RMC algorithm are presented. The code is written in FORTRAN 77 and is performed on Intel PARAGON using p processors. Each processor executes the same program for N/p number of trajectories, i.e. - N/p independent realizations of the random variable. At the end the host processor collects the results of all realizations and computes the average value.

Numerical tests are performed for a large number of test matrices - general symmetric sparse matrices and band sparse symmetric matrices.

The test matrices are produced using a specially created generator of symmetric matrices called *MATGEN*. This generator allows to generate matrices with a given size, given sparsity and fixed largest and smallest eigenvalue (fixed condition number). All other eigenvalues are chosen to be randomly distributed. Using MATGEN-program it is also possible to put a gap of a given size into the spectrum of the matrix. For producing such matrices in MATGEN Jacobi rotations are used such

Table 1: **Test matrices**

Name	Size	Non − zero el. per row	λ_{min}	λ_{max}
$m128.52$	128	52	1.0000	64.0000
$m512.178$	512	178	1.0000	64.0000
$m1000.39$	1000	39	−1.9000	1.0000
$m1024.56$	1024	56	1.0000	64.0000
$m1024.322$	1024	322	1.0000	64.0000
$m2000.56$	2000	56	1.0000	64.0000

that the angle of rotation and the place of appearing the non-zero entrances are randomly chosen. The test matrices used in our numerical experiments are of size 128, 512, 1000, 1024, and 2000 and have different number of non-zero elements. Some of the most important parameters of the matrices are shown in Table 1.

The name of matrices contains the size of the matrix and also a parameter which indicates the sparsity. So, we a able to control the parallel behaviors of the algorithm for different levels of sparsity and to study the dependence between the computational time and the size of the matrices.

Some information about the computations complexity, speed-up and parallel efficiency of the algorithm is presented in Tables 2 – 4 as well as on on Figures 1 and 2.

Tables 2 and 3 present results for matrices of different size and sparsity when a **small number** of Monte Carlo iterations is needed to receive a good accuracy. Subtables "a" show the dependence between calculated values and the number of random trajectories. Subtables "b" contain an information about the dependence of the computational complexity, speed-up and parallel efficiency from the used number of processors p of Intel PARAGON.

Table 4 presents results for a matrix of size 1024 with a given sparsity (322 non-zero elements per row) when a **large** number of iterations are needed ($m = 129$ for Table 4). Here an information about the computational error, time, speed-up and parallel efficiency is given. The results for the parallel efficiency are not very good, because the computational time is very small when a large number of processors is used. In fact, the computational time is comparable with the time needed for communications. It means that even for matrices of large size (up to 2000×2000) it is not necessary to use a lot of processors. Our results show that the parallel efficiency increases when the number of random trajectories increases.

Our numerical tests performed for a large number of matrices confirm the independence of the algorithmic complexity from the size of the matrix. The results presented on Table 5 show that:

Table 2: **Resolvent Monte Carlo Method (RMC) for m512.52** (λ_{min} = 1.00000).

a) The solution when the number of trajectories increases.

Number of traject.	10^3	10^4	10^5	10^6
Calculated λ_{min}	1.0278	0.9958	0.999984	1.000010

b) Implementation on PARAGON (Num. of tr. $N = 10^6$). Calculated $\lambda_{min} = 1.000010$.

Numb. nodes	1	2	3	4	5	6	7	8	9	10
Time (s)	70.75	35.97	24.9	18.3	14.78	12.96	11.49	9.63	8.67	7.68
Speedup S	1	1.97	2.84	3.86	4.78	5.46	6.16	7.34	8.16	9.21
Eff. E	1	0.98	0.95	0.97	0.96	0.91	0.88	0.92	0.91	0.92

Table 3: **Resolvent Monte Carlo Method (RMC) for m1000.39** ($\lambda_{min} = -1.9000$).

a) The solution when the number of trajectories increases.

Number of traject.	10^3	10^4	10^5	10^6
Calculated λ_{min}	-1.9068	-1.9011	-1.90021	-1.900057

b) Implementation on PARAGON (Num. of tr. $N = 10^6$). Calculated $\lambda_{min} = -1.900057$.

Numb. nodes	1	2	3	4	5	6	7	8	9	10
Time (s)	26.72	14.05	10.05	7.97	6.75	5.02	4.51	3.75	3.36	3.30
Speedup S	1	1.90	2.66	3.35	3.96	5.32	5.921	7.13	7.95	8.11
Eff. E	1	0.95	0.89	0.84	0.79	0.89	0.84	0.89	0.88	0.81

Table 4: **Resolvent Monte Carlo Method (RMC) for m1024.322** ($\lambda_{min} = 1.0$). **Parameters of the problem: M=129, L=5, f - unit vector.**

a) The solution when the number of trajectories increases.

$Number of$ $traject.$	10^3	10^4	10^5	10^6
$Calculated$ λ_{min}	0.9268	0.9865	0.9935	0.9959

b) Implementation on PARAGON (Num. of tr. $N = 10^5$).

$Nnodes$	λ_{min}	$Time(s)$	$Speed-up$	$Efficiency$
1	0.9935	12.896	1	1
2	0.9935	6.896	1.870	0.935
3	0.9954	4.736	2.722	0.907
4	0.9923	3.648	2.722	0.680
5	0.9946	3.616	3.535	0.707
6	0.9966	3.648	3.566	0.707
7	0.9931	3.552	3.535	0.594
8	0.9935	3.104	3.630	0.505
9	0.9964	3.008	4.154	0.453
10	0.9945	2.880	4.287	0.461

Table 5: **Computing time and matrix size**

a) Rezults for matrix 128×128. Number of nonzero elements = 6714.
Exact $\lambda_{max} = 64.0$. Parameters: M=47, L=7.

$Number$ $traject.$	$Calculated$ λ_{max}	$Time$
1000	64.1350	0.256
10000	63.3300	2.112
100000	63.1843	21.600
1000000	63.189	208.256

b) Rezults for matrix 1000×1000. Number of nonzero elements = 38748.
Exact $\lambda_{min} = 1.0$. Parameters: M=47, L=7.

$Number$ $traject.$	$Calculated$ λ_{max}	$Time$
1000	0.9981	0.128
10000	0.9997	1.344
100000	1.000051	13.184
1000000	1.000033	132.288

c) Rezults for matrix 1024×1024. Number of nonzeroelements = 57538.
Exact $\lambda_{min} = 64.0$. Parameters: M=17, L=5.

$Number$ $traject.$	$Calculated$ λ_{max}	$Time$
1000	64.3462	0.256
10000	64.1883	1.856
100000	64.1724	18.176
1000000	64.1699	181.504

d) Rezults for matrix 2000×2000. Number of nonzero elements = 112594.
Exact $\lambda_{min} = 64.0$. Parameters: M=17, L=5.

$Number$ $traject.$	$Calculated$ λ_{max}	$Time$
1000	63.9943	0.192
10000	64.0050	1.408
100000	64.0204	13.312
1000000	64.0265	133.248

Table 6: **Resolvent Monte Carlo Method (RMC) for m1000.39**
Exact $\lambda_{max} = 1.0$.

L	M	$Calculated$ λ_{max}	$Time$
2	2	1.0921	0.064
2	5	1.0833	0.064
2	10	1.0773	0.064
2	20	1.0729	0.064

L	M	$Calculated$ λ_{max}	$Time$
5	2	1.0658	0.128
5	5	1.0338	0.128
5	10	1.0191	0.128
5	20	1.0133	0.128

L	M	$Calculated$ λ_{max}	$Time$
10	2	1.0599	0.192
10	5	1.0195	0.192
10	10	1.0040	0.192
10	20	0.9983	0.192
10	30	0.9956	0.192

Remarks: 1. The number of trajectories N is 1000.
2. The time is measured during the program implementation on Intel PARAGON (1 processor).

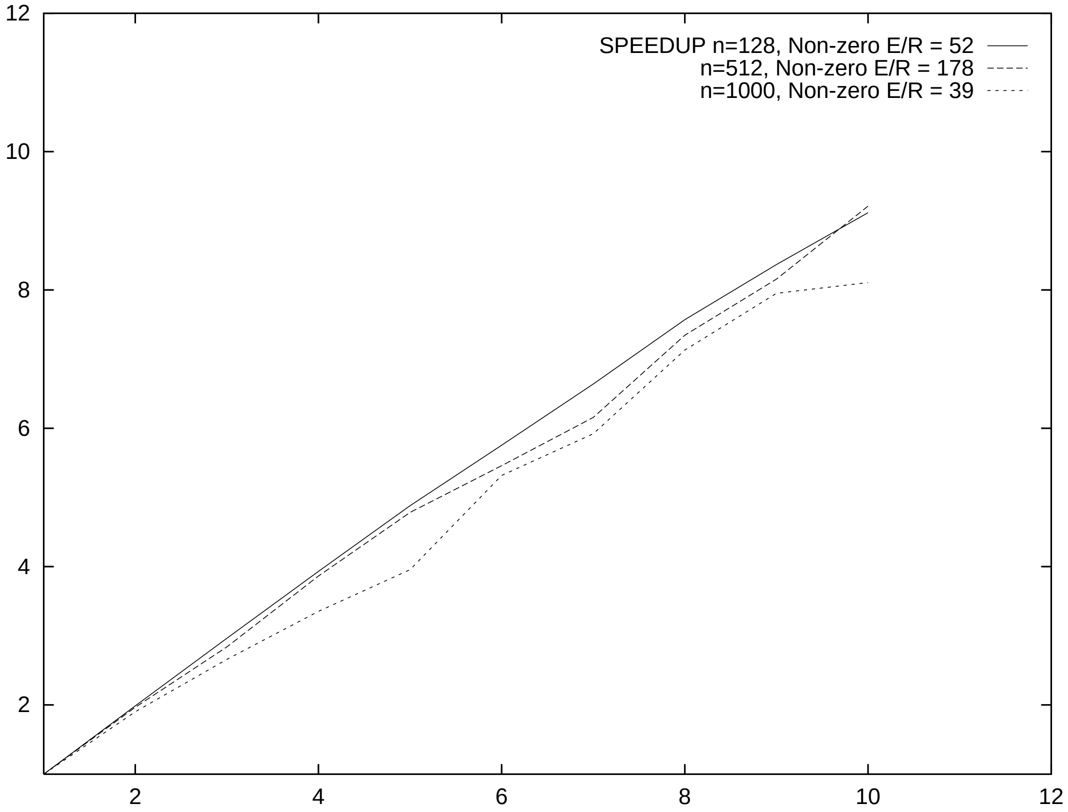

Figure 1
Dependence of the Speedup from the number of processors for different matrices (the computational time is relatively large in comparison with communicational time for all matrices; the number of realizations N for all matrices is relatively large).

- the computational time is almost independent of the size of the matrix;
- a linear dependence between the computational time and the number of the random trajectories is observed;
- a linear dependence between the computational time and the mean value of the number of non-zero entries of each row of the matrix is realized.

Some numerical results showing the dependence of the accuracy from the parameters m and l are presented on Table 6.

Figures 1-2 present some results for speed-up when different numbers of processors of Intel PARAGON machine are used. One can see, that when the number of the random trajectories is large the speed-up is almost linear and it is closed to the best value of the speed-up, i.e. to the value of p (Figure 1). When the number of processors is small (with respect to the computational complexity) the speed-up is linear. If the number of processors p is large and the computational time is small the speed-up is not so good (see, Figure 2). It may happen for tasks with a very small computational

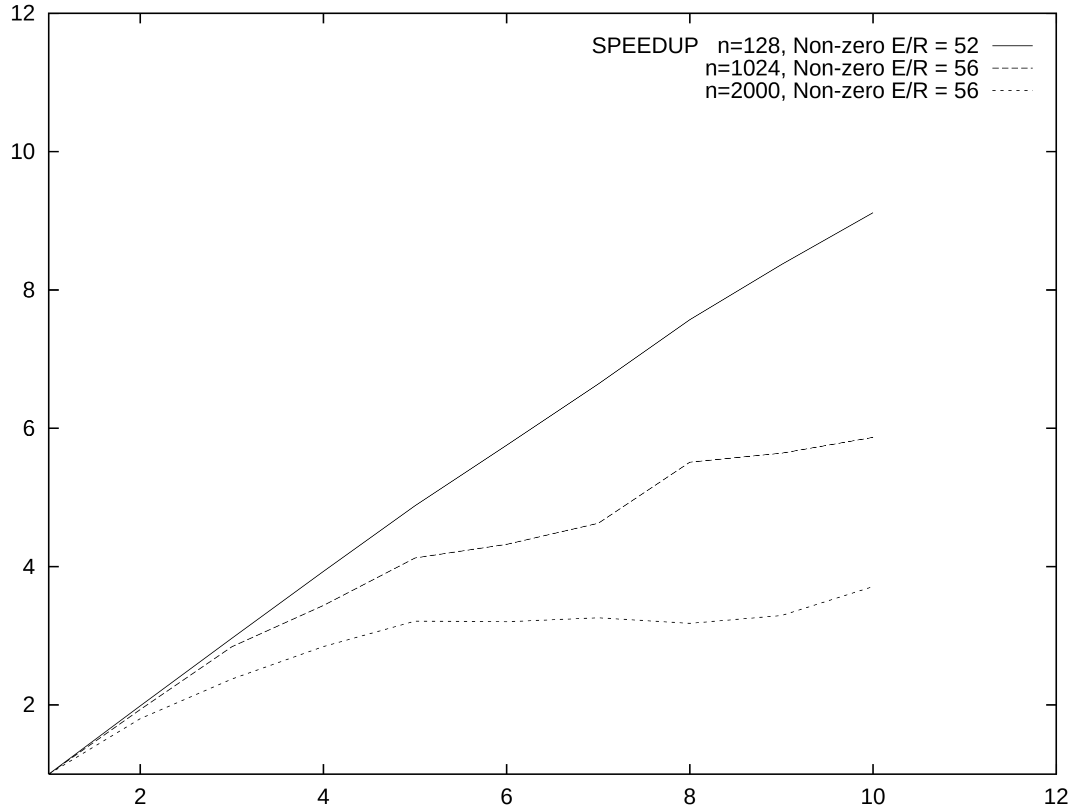

Figure 2
Dependence of the Speedup from the number of processors for different matrices (the computational time is relatively small in comparison with communicational time for matrices of size 1024 and 2000 since the number of realizations N is 10 times smaller than in the case of matrix m128.52).

complexity when such large number of processors is not needed. The case of small computational complexity may happen not only because of small number of non-zero elements per row, but also because of small number of realizations N needed for realizing the algorithm. The last case is shown on Figure 2.

3.4 Concluding Remarks

A parallel RMC algorithm for calculating eigenvalues is presented and studied in this paper. Estimators for the speedup as well as for parallel efficiency are considered.

The algorithm under consideration is *almost optimal* from statistical point of view, i.e the variance of the random variable, which is equal to the eigenvalue is *almost optimal* in the meaning of definition given in [Di91].

The convergence of RMC algorithm depends on the spectrum of the matrix. The systematic error is

$$O\left[\left(\frac{2\lambda_1+\lambda_n}{2\lambda_1+\lambda_{n-1}}\right)^m\right], \tag{21}$$

where m is the power (or the number of iterations).

When $\lambda_n \approx -2\lambda_1$ the convergence is very good. It is clear from (21) that for a positive or negative defined matrices the convergence decreases, so that the best convergence which can be reached is $O[(2/3)^m]$. The studied algorithm has strong requirements about matrices for which it can be applied: the error from the power method used on the resolvent matrix determines the value of the of the parameter m; the error coming from the representation of the resolvent matrix as a series determines the parameter l. The values of m and l are not independent, since they determine the binomial coefficients C^l_{m+l-1} which grow exponentially with l.

The results obtained by RMC for sparse matrices show that:

- the computational time is almost independent from the size of the matrix;
- there is a linear dependence between the computational time and the number of the random trajectories;
- there is a linear dependence between the computational time and the mean value of the number of the non-zero entrances of each row of the matrix;
- the speedup is almost linear when the computational time for every processor is not too small.

For matrices A with "well distributed" eigenvalues, when the parameter q is "not too large" the RMC algorithm works well. When the largest eigenvalue of the resolvent matrix $(I-qA)^{-1}$ is "well isolated" a small number of the power "m" is needed ($m = 2,\dots,5$). In this case the number of moves in every Markov chain can be also small ($l = 5,\dots,10$) and the accuracy of calculations is high. In this case the results are not very sensitive to the value of the parameter q, which controls the speed of convergence of the Monte Carlo iterative procedure. When $m > 35$ the results are sensitive to q. This case takes place when $\lambda_1((I-qA)^{-1}/\lambda_2((I-qA)^{-1}$ is approximately equal to 1. In this case it is better to apply the Inverse Monte Carlo iterative algorithm.

4 Acknowledgments

This work was partially supported by the Ministry of Science, Education and Technology of Bulgaria under grant I 501/95 as well as under INCO-Copernicus Project: 960237 (STABLE) of the Commission of the European Communities. Authors are grateful to professor Bernard Philippe - coordinator of the project "STABLE" for the very helpful discussions of the results presented here.

References

[Di91] Dimov I.T., *Minimization of the probable error forsome Monte Carlo methods* - Proc. of the Summer School on **Mathematical Modelling and Scientific Computations**, 23-28.09.1990, Albena, Bulgaria, Sofia, Publ. House of the Bulg. Acad. Sci., 1991, pp. 159 – 170.

[DT93] I. Dimov, O. Tonev, *Random walk on distant mesh points Monte Carlo methods*, **Journal of Statistical Physics, vol. 70** (5/6), 1993, pp. 1333 – 1342.

[DT93a] I. Dimov, O. Tonev, *Monte Carlo algorithms: performance analysis for some computer architectures.* **Journal of Computational and Applied Mathematics, vol. 48** (1993) pp. 253 – 277.

[Ka50] H. Kahn, *Random sampling (Monte Carlo) techniques in neutron attenuation problems*, **Nucleonics , vol. 6**, No 5 (1950), pp. 27 – 33; **vol. 6**, No 6 (1950), pp. 60 – 65.

[MAD94] G. Megson, V. Aleksandrov, I. Dimov, *Systolic Matrix Inversion Using a Monte Carlo Method*, **Journal of Parallel Algorithms and Appl., vol. 3, No 3/4** (1994), pp. 311 – 330.

[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).

[Sa89] K. Sabelfeld, *Algorithms of the Method Monte Carlo for Solving Boundary Value Problems* , **Nauka**, Moscow, 1989.

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

[Vl56] 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.