Monte Carlo Numerical Treatment of Large Linear Algebra Problems*

Ivan Dimov, Vassil Alexandrov, Rumyana Papancheva, and Christian Weihrauch

Centre for Advanced Computing and Emerging Technologies
School of Systems Engineering, The University of Reading
Whiteknights, PO Box 225, Reading, RG6 6AY, UK
{i.t.dimov, v.n.alexandrov, c.weihrauch}@reading.ac.uk
Institute for Parallel Processing, Bulgarian Academy of Sciences
Acad. G. Bonchev 25 A, 1113 Sofia, Bulgaria
ivdimov@bas.bg, rumi@parallel.bas.bg

Abstract. In this paper we deal with performance analysis of Monte Carlo algorithm for large linear algebra problems. We consider applicability and efficiency of the Markov chain Monte Carlo for large problems, i.e., problems involving matrices with a number of non-zero elements ranging between one million and one billion. We are concentrating on analysis of the almost Optimal Monte Carlo (MAO) algorithm for evaluating bilinear forms of matrix powers since they form the so-called Krylov subspaces.

Results are presented comparing the performance of the Robust and Non-robust Monte Carlo algorithms. The algorithms are tested on large dense matrices as well as on large unstructured sparse matrices.

Keywords: Monte Carlo algorithms, large-scale problems, matrix computations, performance analysis, iterative process.

1 Introduction

Under *large* we consider problems involving dense or general sparse matrices with a number of non-zero elements ranging between one million and one billion.

It is known that Monte Carlo methods give statistical estimates for bilinear forms of the solution of systems of linear algebraic equations (SLAE) by performing random sampling of a certain random variable, whose mathematical expectation is the desired solution [8]. The problem of variance estimation, in the optimal case, has been considered for extremal eigenvalues [7,9]. In [5,6] we analyse the errors of iterative Monte Carlo for computing bilinear forms of matrix powers. If one is interested to apply Markov chain Monte Carlo for large

* Partially supported by the Bulgarian Ministry of Education and Science, under grant I-1405/2004. The authors would like to acknowledge the support of the European Commission's Research Infrastructures activity of the Structuring the European Research Area programme, contract number RII3-CT-2003-506079 (HPC-Europa).

Y. Shi et al. (Eds.): ICCS 2007, Part I, LNCS 4487, pp. 747–754, 2007.

problems, then the applicability and robustness of the algorithm should be studied. The run of large-scale linear algebra problems on parallel computational systems introduce many additional difficulties connected with data parallelization, distribution of parallel subtasks and parallel random number generators. But at the same time one may expect that the influence of unbalancing of matrices is a lot bigger for smaller matrices (of size 100, or so) (see [5]) and with the increasing the matrix size robustness is also increasing. It is reasonable to consider large matrices, and particularly large unstructured sparse matrices since such matrices appear in many important real-live computational problems.

We are interested in the bilinear form of matrix powers since it is a basic subtask for many linear algebra problems:

$$(v, A^k h). \tag{1}$$

If x is the solution of a SLAE $Bx = b$, then

$$(v, x) = \left(v, \sum_{i=0}^{k} A^i h \right),$$

where the Jacobi Over-relaxation Iterative Method has been used to transform the SLAE into the problem $x = Ax + h$.

For an arbitrary large natural number k the Rayleigh quotient can be used to obtain an approximation for λ_1, the dominant eigenvalue, of a matrix A:

$$\lambda_1 \approx \frac{(v, A^k h)}{(v, A^{k-1} h)}.$$

In the latter case we should restrict our consideration to real symmetric matrices in order to deal with real eigenvalues. Thus it is clear that having an efficient way of calculating (1) is important. This is especially important in cases where we are dealing with large matrices.

2 Markov Chain Monte Carlo

The algorithm we use in our runs is the so-called Almost Optimal Monte Carlo (MAO) algorithm studied in [1,3,4]. We consider a Markov chain $T = \alpha_0 \to \alpha_1 \to \alpha_2 \to \ldots \to \alpha_k \to \ldots$ with n states. The random trajectory (chain) T_k of length k starting in the state α_0 is defined as follows: $T_k = \alpha_0 \to \alpha_1 \to \ldots \to \alpha_j \to \ldots \to \alpha_k$, where α_j means the number of the state chosen, for $j = 1, \ldots, k$. Assume that $P(\alpha_0 = \alpha) = p_\alpha$, $P(\alpha_j = \beta | \alpha_{j-1} = \alpha) = p_{\alpha\beta}$, where p_α is the probability that the chain starts in state α and $p_{\alpha\beta}$ is the transition probability to state β after being in state α. Probabilities $p_{\alpha\beta}$ define a transition matrix P.

In all algorithms used in this study we will consider a special choice of density distributions p_i and p_{ij} defined as follows:

$$p_i = \frac{|v_i|}{\| v \|}, \quad \| v \| = \sum_{i=1}^{n} |v_i| \quad \text{and} \quad p_{ij} = \frac{|a_{ij}|}{\| a_i \|}, \quad \| a_i \| = \sum_{j=1}^{n} |a_{ij}|. \tag{2}$$

The specially defined Markov chain induces the following product of matrix/vector entrances and norms:

$$A_v^k = v_{\alpha_0} \prod_{s=1}^{k} a_{\alpha_{s-1}\alpha_s}; \quad \| A_v^k \| = \| v \| \times \prod_{s=1}^{k} \| a_{\alpha_{s-1}} \| .$$

We have shown in [5] that the value

$$\bar{\theta}^{(k)} = \frac{1}{N} \sum_{i=1}^{N} \theta_i^{(k)} = sign\{A_v^k\} \| A_v^k \| \frac{1}{N} \sum_{i=1}^{N} \{h_{\alpha_k}\}_i \tag{3}$$

can be considered as a MC approximation of the form $(v, A^k h)$. For the probability error of this approximation one can have:

$$R_N^{(k)} = \left| (v, A^k h) - \bar{\theta}^{(k)} \right| = c_p \sigma\{\theta^{(k)}\} N^{-\frac{1}{2}},$$

where c_p is a constant.

In fact, (3) together with the sampling rules using probabilities (2) defines the MC algorithm used in our runs. Naturally, the quality of the MC algorithm depends on the behaviour of the standard deviation $\sigma\{\theta^{(k)}\}$. So, there is a reason to consider a special class of *robust MC algorithms*. Following [5] under *robust MC algorithms* we assume algorithms for which the standard deviation does not increase with the increasing of the matrix power k. So, robustness in our consideration is not only a characteristic of the quality of the algorithm. It also depends on the input data, i.e., on the matrix under consideration. As better balanced is the iterative matrix, and as smaller norm it has, as bigger a chances to get a robust MC algorithm.

3 Numerical Experiments

In this section we present results on experimental study of quality of the Markov chain Monte Carlo for large matrices. We run algorithms for evaluating mainly bilinear forms of matrix powers as a basic Monte Carlo iterative algorithms as well as the algorithm for evaluating the dominant eigenvalue. In our experiments we use dense and unstructured sparse matrices of sizes

- $n = 1000$, $n = 5000$, $n = 10000$, $n = 15000$, $n = 20000$, $n = 40000$.

We can control some properties of random matrices. Some of the matrices are well balanced (in some sense matrices are *close* to stochastic matrices), some of them are not balanced, and some are completely unbalanced. Some of the iterative matrices are with small norms which makes the Markov chain algorithm robust (as we showed in Section 2), and some of the matrices have large norms. Since the balancing is responsible for the variance of the random variable dealing with unbalanced matrices we may expect higher values for the stochastic error.

In such a way we can study how the stochastic error propagates with the number of Monte Carlo iterations for different matrices. Dealing with matrices of small norms we may expect high robustness of the Markov chain Monte Carlo and a high rate of convergence. To be able to compare the accuracy of various runs of the algorithm for computing bilinear forms of matrix powers we also compute them by a direct deterministic method using double precision. These runs are more time consuming since the computational complexity is higher than the complexity for Markov chain Monte Carlo, but we accept the results as *"exact results"* and use them to analyse the accuracy of the results produced by our Monte Carlo code.

For sparse matrices we use the Yale sparse matrix format [2]. We exploit the sparsity in sense that the used almost optimal Monte Carlo algorithm only deals with non-zero matrix entrances. The Yale sparse matrix format is very suitable since is allows to present large unstructured sparse matrices in a compact form in the processor's memory [2]. The latter fact allows to perform *jumps* of the Markov chain from one to another non-zero elements of a given matrix very fast.

We also study the scalability of the algorithms under consideration. We run our algorithms on different computer systems. The used systems are given below:

- IBM p690+ Regatta system cluster of IBM SMP nodes, containing a total of 1536 IBM POWER5 processors;
- Sun Fire 15K server with 52x0,9GHz UltraSPARC III processors;
- SGI Prism equipped with 8 x 1.5 GHz Itanium II processors and 16 GByte of main memory.

On Figure 1 we present results for the Monte Carlo solution of bilinear form of a dense matrix of size $n = 15000$ from the matrix power k (the matrix power corresponds to the number of moves in every Markov chain used in computations). The Monte Carlo algorithm used in calculations is robust. For comparison we present *exact* results obtained by deterministic algorithm with double precision. One can not see any differences on this graph. As one can expect the error of the robust Monte Carlo is very small and it decreases with increasing the matrix power k. In fact the stochastic error exists but it increases rapidly with increasing of k. This fact is shown more precisely on Figure 2.

The Monte Carlo probability error $R_N^{(k)}$ and the Relative Monte Carlo probability error $Rel_N^{(k)}$ was computed in the following way:

$$R_N^{(k)} = \left| \frac{1}{N} \sum_{i=1}^{N} \theta_i^{(k)} - \frac{(v, A^k h)}{(v, h)} \right|, \quad Rel_N^{(k)} = \frac{(v, h)}{(v, A^k h)} R_N^{(k)}.$$

In the robust case the relative MC error decreases to values smaller than 10^{-22} when the number of MC iterations is 20, while for in the non-robust case the corresponding values slightly increase with increasing of matrix power (for $k = 20$ the values are between 10^{-3} and 10^{-2} (see Figure 2).

If we apply both the robust and the non-robust Markov chain Monte Carlo to compute the dominant eigenvalue of the same dense matrices of size $n = 15000$ then we get the result presented on Figure 3.

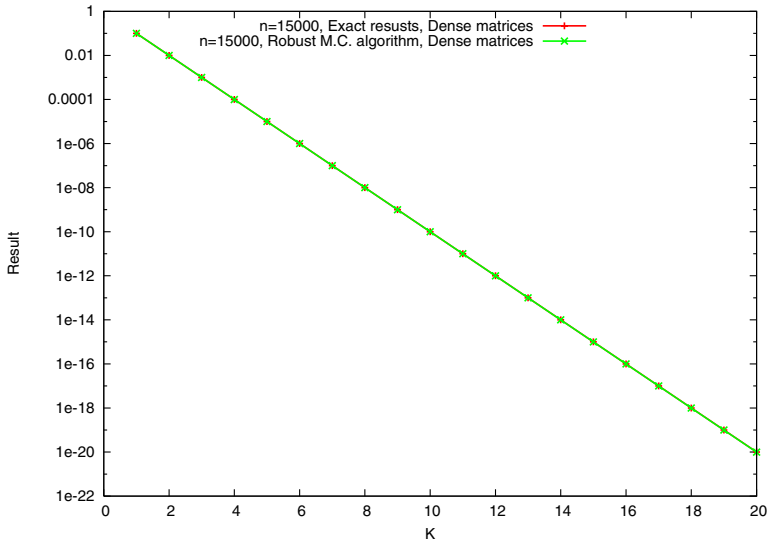

Fig. 1. Comparison of Robust Monte Carlo algorithm results with the *exact* solution for the bilinear form of a dense matrix of size $n = 15000$

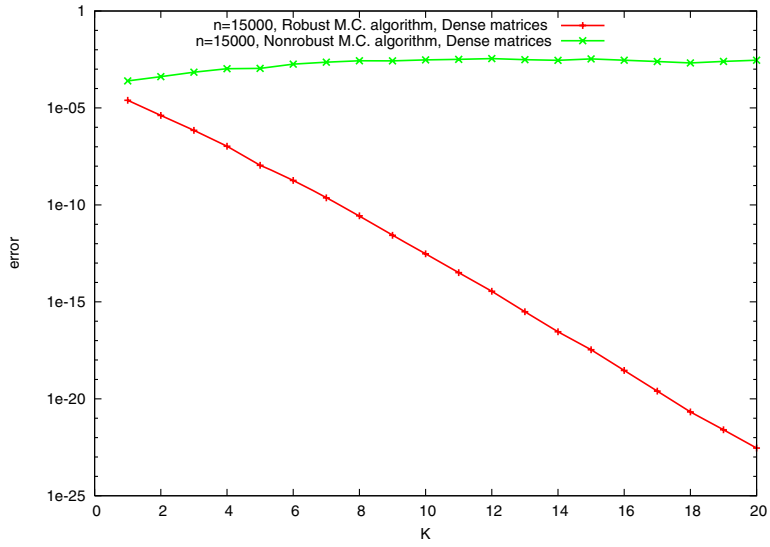

Fig. 2. Comparison of the Relative MC error for the robust and non-robust algorithms. Matrices of size $n = 15000$ are used.

We should stress on the fact that the random matrices are very similar. Only the difference is that in the robust case the matrix is well balanced. From Figure 3 one can see that the oscillations of the solution are much smaller when the matrix

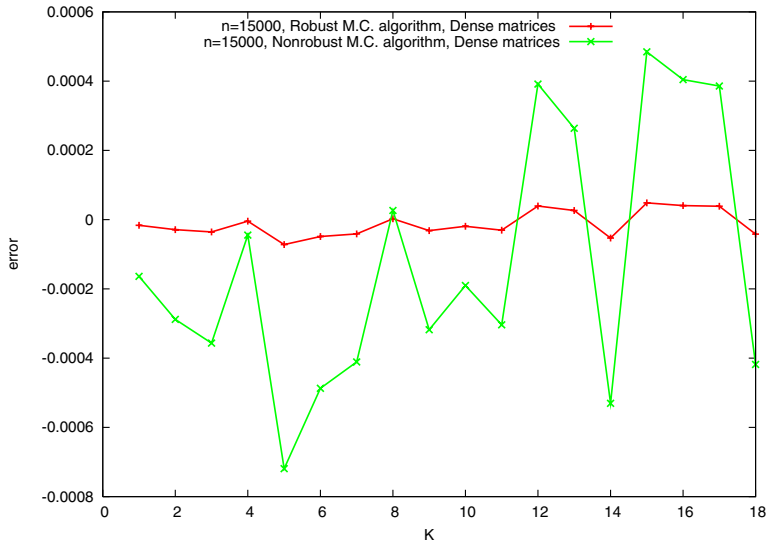

Fig. 3. The relative MC error for the robust and non-robust algorithms. The matrix size is $n = 15000$.

is well balanced. The reason for that is that the variance for the well balanced matrix is much smaller than for non-balanced matrix.

Very similar results are obtained for matrices of size $1000, 5000, 10000, 20000$, and 40000. Some results for sparse matrices are plotted on Figures 4, and 5.

Results of Monte Carlo computation of the bilinear form of an unstructured sparse matrix of size 10000 are plotted on Figure 4. The MC results are compared with the *exact* results. On this graph one can not find any differences between MC results and the *exact* solution. One can see that if the robust algorithm is applied for solving systems of linear algebraic equations or for computing the dominant eigenvalue of real symmetric matrices (in order to get real eigenvalues), then just 5 or 6 Monte Carlo iterations are enough to get fairly accurate solution (with 4 right digits). If we present the same results for the same sparse matrix in a logarithmic scale, then one can see that after 20 iterations the relative MC error is smaller than 10^{-20} since the algorithm is robust and with increasing the number of iterations the stochastic error decreases dramatically. Similar results for a random sparse matrix of size 40000 are shown on Figure 5.

Since the algorithm is robust and the matrix is well balanced the results of MC computations are very closed to the results of deterministic algorithm performed with a double precision.

Our observation from the numerical experiments performed are that the error increases linearly if k is increasing. The larger the matrix is, the smaller the influence of non-balancing is. This is also an expected result since the stochastic error is proportional to the standard deviation of the random variable computed as a weight of the Markov chain. When a random matrix is very large it is becoming

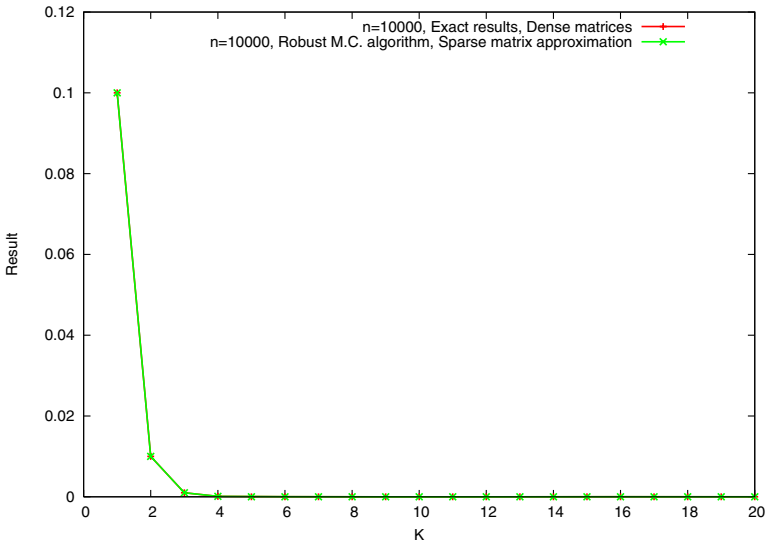

Fig. 4. Comparison of the MC results for bilinear form of matrix powers for a sparse matrix of size $n = 10000$ with the exact solution. 5 or 6 Monte Carlo iterations are enough for solving the system of linear algebraic equations or for computing the dominant eigenvalue for the robust Monte Carlo.

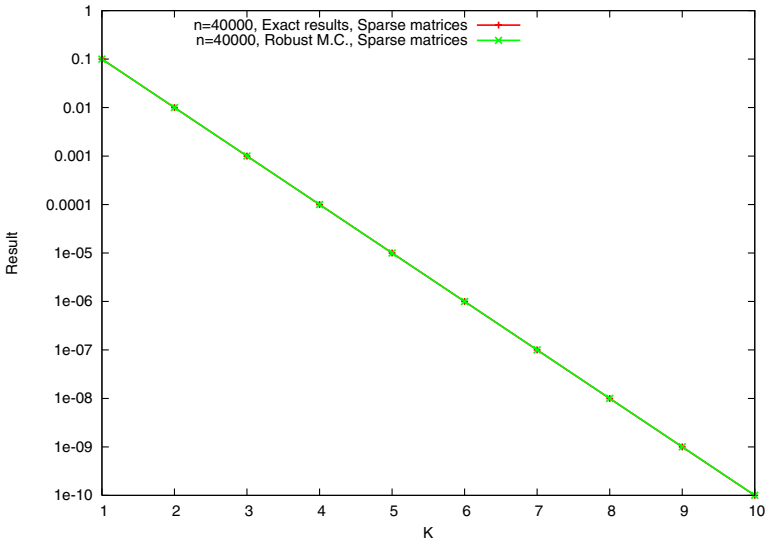

Fig. 5. Comparison of the MC results for bilinear form of matrix powers for a sparse matrix of size $n = 40000$ with the exact solution

more close (in some sense) to the stochastic matrix and the standard deviation of the random variable statistically decreases which statistically increases the accuracy.

4 Conclusion

In this paper we have analysed the performance of the proposed MC algorithm for linear algebra problems. We are focused on the computing bilinear form of matrix powers $(v, A^k h)$ as a basic subtask of MC algorithms for solving a class of Linear Algebra problems. We study the applicability and robustness of Markov chain Monte Carlo. The robustness of the Monte Carlo algorithm with large dense and unstructured sparse matrices has been demonstrated. It's an important observation that the balancing of the input matrix is very important for MC computations since it decreases the stochastic error and improved the robustness.

References

1. V. Alexandrov, E. Atanassov, I. Dimov: *Parallel Quasi-Monte Carlo Methods for Linear Algebra Problems*, Monte Carlo Methods and Applications, Vol. 10, No. 3-4 (2004), pp. 213-219.
2. R. E. Bank and C. C. Douglas: *Sparse matrix multiplication package (SMMP)*, Advances in Computational Mathematics, Vol. 1, Number 1 / February (1993), pp. 127-137.
3. I. Dimov: *Minimization of the Probable Error for Some Monte Carlo methods*. Proc. Int. Conf. on Mathematical Modeling and Scientific Computation, Albena, Bulgaria, Sofia, Publ. House of the Bulgarian Academy of Sciences, 1991, pp. 159-170.
4. I. Dimov: *Monte Carlo Algorithms for Linear Problems*, Pliska (Studia Mathematica Bulgarica), Vol. 13 (2000), pp. 57-77.
5. I. Dimov, V. Alexandrov, S. Branford, and C. Weihrauch: *Error Analysis of a Monte Carlo Algorithm for Computing Bilinear Forms of Matrix Powers*, Computational Science (V.N. Alexandrov et al. Eds.), Lecture Notes in Computing Sciences, Springer-Verlag Berlin Heidelberg, Vol. 3993, (2006), 632-639.
6. C. Weihrauch, I. Dimov, S. Branford, and V. Alexandrov: *Comparison of the Computational Cost of a Monte Carlo and Deterministic Algorithm for Computing Bilinear Forms of Matrix Powers*, Computational Science (V.N. Alexandrov et al. Eds.), Lecture Notes in Computing Sciences, Springer-Verlag Berlin Heidelberg, Vol. 3993, (2006), 640-647.
7. I. Dimov, A. Karaivanova: *Parallel computations of eigenvalues based on a Monte Carlo approach*, Journal of Monte Carlo Method and Applications, Vol. 4, Nu. 1, (1998), pp. 33-52.
8. J.R. Westlake: *A Handbook of Numerical matrix Inversion and Solution of Linear Equations*, John Wiley & Sons, inc., New York, London, Sydney, 1968.
9. M. Mascagni, A. Karaivanova: *A Parallel Quasi-Monte Carlo Method for Computing Extremal Eigenvalues*, Monte Carlo and Quasi-Monte Carlo Methods (2000), Springer, pp. 369-380.