BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **17**, No 5
Special issue with selected papers from the workshop
"Two Years Avitohol: Advanced High Performance Computing Applications 2017"

Sofia • 2017 Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.1515/cait-2017-0054

On the Monte Carlo Matrix Computations on Intel MIC Architecture

Aneta Karaivanova[1], *Vassil Alexandrov*[2], *Todor Gurov*[1], *Sofiya Ivanovska*[1]

[1]*Institute of Information and Communication Technologies, Bulgarian Academy of Sciences, 1113 Sofia, Bulgaria*
[2]*Barcelona Supercomputing Centre* (*BSC*), *Barcelona, Spain, and ICREA – Catallan Institution for Advanced Research Studies, Spain*
E-mails: anet@parallel.bas.bg gurov@parallel.bas.bg sofia@parallel.bas.bg vassil.alexandrov@bsc.es

Abstract: *The tightened energy requirements when designing state-of-the-art high performance computing systems lead to the increased use of computational accelerators. Intel introduced the Many Integrated Core* (*MIC*) *architecture for their line of accelerators and successfully competes with NVIDIA on basis of price/performance and ease of development. Although some codes may be ported successfully to Intel MIC architecture without significant modifications, in order to achieve optimal performance one has to make the best use of the vector processing capabilities of the architecture. In this work we present our implementation of Quasi-Monte Carlo methods for matrix computations specifically optimised for the Intel Xeon Phi accelerators. To achieve optimal parallel efficiency we make use of both MPI and OpenMP.*

Keywords: *Monte Carlo, matrix computations, Intel MIC, High Performance Computing* (*HPC*) *scalability.*

1. Introduction

First proposed by von Neumann and Ulam, Monte Carlo Methods (MCMs) for solving linear algebra problems have been known since the middle of the last century. They give statistical estimates for the elements of the inverse of a matrix or for components of the solution vector of a linear system by performing random

sampling of a certain random variable whose expected value is the desired solution. Perhaps the first application of MCMs in linear algebra appeared in a paper by Forsythe and Leibler [16] in 1950. In the following years significant contributions were made, especially by Wasow [30], Curtiss [10], Halton [19], Hammersley and Handscomb [18] and Sobol [26]. These methods were recognized as useful in the following situations [29]: when obtaining a quick rough estimate of solution, which will then be refined by other methods; when the problem is too large or too intricate for any other treatment; when just one component of the solution vector or one element of the inverse matrix is desired.

There has been renewed interest in MCMs since the 90-ties, for example [7, 9-11, 13, 18, 20, 21, 26, 27, 31, 32] and many others, the primary reason for this is the efficiency of parallel MCMs in the presence of high communication costs. The second reason for the recent interest in MCMs is that the methods have evolved significantly since the early days.

Much of the effort in the development of Monte Carlo methods has been in the construction of variance reduction techniques which speed up the computation by reducing the rate of convergence of crude MCM, which is $O(N^{-1/2})$. An alternative approach to acceleration is to change the type of random sequence, and hence improve the behaviour with N. Quasi-Monte Carlo methods (QMCMs) use quasirandom (also known as low-discrepancy) sequences [4, 5] instead of pseudorandom sequences, with the resulting convergence rate for numerical integration being as good as $O((\log N)^k/N)$. The first results of using QMCMs for linear algebra problems were presented by Mascagni and Karaivanova (see for example [24, 25]) and Halton [19, 20].

One of the interesting MC applications for solving systems of linear algebraic equations, developed recently by V. N. Alexandrov (see [1-3]) and his group, is building of sparse approximate inverse preconditioners based on a Monte Carlo method for computing the inverse matrix (see for example [1, 3, 15, 28]). The Monte Carlo preconditioner outperforms the deterministic preconditioner based on minimization of the Frobenius norm in terms of computing time and applicability. The comparisons of parallel versions of the approaches confirm the advantages of Monte Carlo in building of sparse approximate inverse preconditioner. Later, a Quasi-Monte Carlo approach has been developed. The results with Quasi-Monte Carlo algorithm based on Sobol's sequence are slightly better than those obtained by Monte Carlo [2]. The problem arose with the parallel implementation of these algorithms on supercomputers with Intel MIC architecture. In order to improve the parallel efficiency we have applied various techniques which are presented in this paper.

2. Background and related research

The Intel's Many Integrated Core (MIC) architecture is used in the Intel Xeon Phi line of processors, which are used as co-processor cards in the first generation, but can be used as fully functional main processors in the subsequent editions. In our current high-performance computing system they are used as co-processors in

servers equipped with standard Intel Xeon CPUs. Even with this limited functionality the Intel Xeon Phis compete with GPU cards for the role of accelerators in heterogeneous systems where they can significantly improve the computational efficiency and power consumption with respect to those resulting from use of standard CPUs. Details on how to program for the Intel Xeon Phi coprocessors can be found in [33]. Some of their main characteristics are: 1) equipped with vector units that allow processing of several integer or floating point numbers at once; 2) running many cores at low frequencies; 3) availability of hyperthreading. In practice, if the vector instructions on such accelerators are not used, then they perform as slow as regular CPUs. The Intel compilers available in the Parallel Studio XE package provide direct access to the vector instructions via compiler intrinsics, thus facilitating the use of vector instructions by the program developers. Another possibility is the direct coding of assembly instructions inside C codes, which is more complex but still feasible for Xeon Phi accelerators. The different ways of using Intel MIC accelerators and their inherent composite structure lead to multiple parallelization approaches that can be applied to such systems. When one tries to implement then in practice, various steps have to be chosen and the particular target system guides these choices.

For our tests we used the Avitohol High Performance System, built with Xeon Phi 7120P accelerators, hosted at our institute. Avitohol consists of 150 servers SL250S equipped with both dual Xeon CPU E5-2650 V2 at 2.60 GHz and dual Xeon Phi 7201P accelerator cards. The total accessible RAM on the system by the regular CPUs and the accelerator cards are 9600 GB and 4800 GB, respectively. The operating system on the servers is Red Hat Enterprise Linux, while Intel's own special version of Linux OS (part of the MPSS package) is installed on the accelerators. Currently the exact versions on the servers and for the MPSS are 6.7 and 3.6-1, respectively. This system achieved 332th place in the Top 500 list when it entered operation, with a theoretical peak performance of about 413 TElop/s, of which 90% is contributed by the accelerators. One can conclude that the optimal use of accelerators is the only way to fully leverage the power of such kinds of systems. However, many software packages do not have optimised versions for accelerators.

Our results on efficient implementation on Avitohol of algorithms for quasirandom sequences generation and of Monte Carlo and Quasi-Monte Carlo algorithms for solving multidimensional integrals can be found in [4, 5, 25], etc. Here we present some results for efficient Monte Carlo matrix computations.

At the end of this section we show the execution time breakdown for the Monte Carlo sparse inverse preconditioner and six test matrices from The University of Florida Sparse Matrix Collection [30] and Matrix market [6], described in Table 1. Figs 1 and 2 show the large communication time which was the motivation for our developments presented in this paper. These developments can be applied for all Monte Carlo algorithms for matrix computations. The experiments below (Fig. 1 and Fig. 2) are run on Marenustrum III Supercomputer at the Barcelona Supercomputer Centre.

Table 1. Test matrices

Matrix	Dimension	No-zeros	Sparsity	Symmetry
Appu	14,000	1,853,104	0.95%	Non-symmetric
Na5	5,832	305,630	0.46%	Symmetric
Nonsym_r5_a11	329,473	10,439,197	0.01%	Non-symmetric
Rdb2048	2,048	12,032	0.29%	Non-symmetric
Sym_r3_a11	20,928	588,601	0.13%	Symmetric
Sym_r4_a11	82,817	2,598,173	0.04%	Symmetric

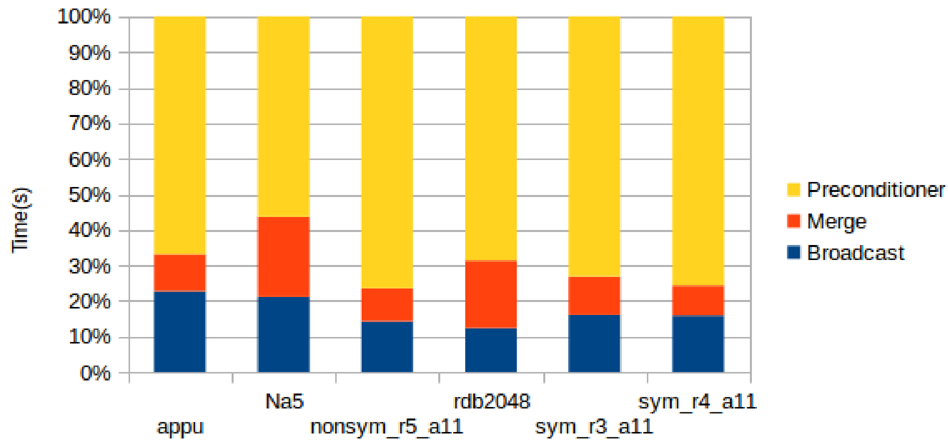

Fig. 1. Execution time breakdown in a 16 cores execution

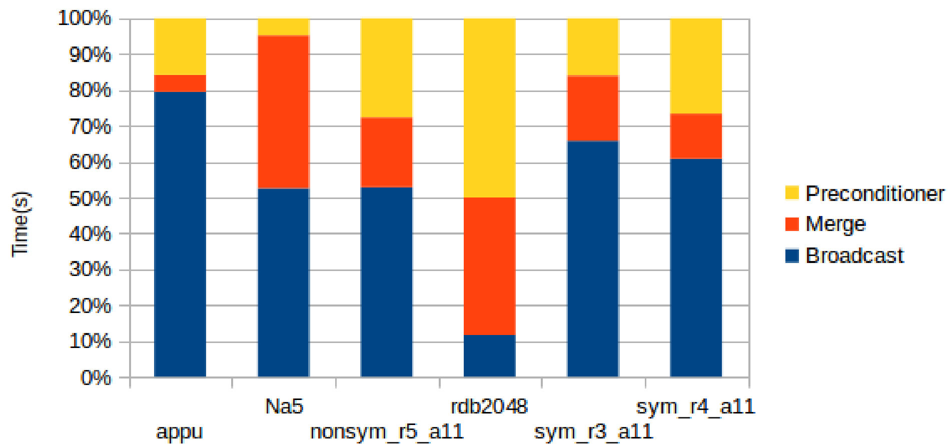

Fig. 2. Execution time breakdown in a 256 cores execution

3. Matrix computations

Monte Carlo methods can be used for various types of linear algebra computations, like finding solutions of a linear system, computing matrix-vector products or estimation of eigenvalues. An approximation to the inverse matrix can also be

computed via Monte Carlo. Most of the Monte Carlo algorithms for solving such problems are based on sampling of appropriate Markov chains. Monte Carlo methods can be used for both dense and sparse matrices. The Markov chain is usually based on jumping along the non-zero elements of the matrix, with transition probability that is proportional to the absolute value of these elements (the so-called Almost Optimal Monte Carlo algorithm (see, e.g., [10, 13, 27]). Depending on the particular problem jumping is done either along rows or columns. For a Quasi-Monte Carlo variant of the same algorithm it is appropriate to consider each Markov chain as corresponding to a different term of the low-discrepancy sequences, and then use different coordinates of this term in order to move along the non-zero elements of the matrix. In both Monte Carlo and Quasi-Monte Carlo algorithms we arrive at the problem of sampling from a discrete distribution. It is important to understand that a naive implementation of the sampling from a discrete distribution would add a factor proportional to the mean number of non-zero elements in a row or column of the matrix in the estimate of number of required operations, thus negating the speed advantage of the Monte Carlo method.

Fortunately, faster methods, that use $O(1)$ number of operations for this sampling exist. Although any such method would be acceptable for the computer implementation of the Monte Carlo algorithm, this is not the case for the corresponding Quasi-Monte algorithm. The reason for this is that the theoretical justification of the Quasi-Monte Carlo method is based on smoothness of the underlying function and in the case of matrix computation this function is a multi-dimensional step function. Any kind of permutation of the indices may result in making this multi-dimensional step function far less smooth, in the sense of increasing its variation. That is why in our programmatic implementation of Quasi-Monte Carlo methods for matrix computations we adopted the tabular method for sampling a discrete distribution, where a suitable k is chosen and the indices of the non-zero elements that correspond to the choice of all numbers of type $r/2^k$ are tabulated. The choice of k is achieved by finding the biggest k such that 2^k is less than the mean number of non-zero elements per row or column.

This approach needs some extra memory, but it is less than that required for the matrix itself, because of the choice of k and the fact that integers are stored instead of double precision numbers. The pre-computing of this table requires number of operations proportional to the total number of non-zero elements of the matrix and thus does not increase the total order of the number of operations.

When we have a pseudo-random number or a coordinate of the low-discrepancy sequence x, and we need to find the corresponding index among a column, we find integer r such that

$$\frac{r}{2^k} \leq x < \frac{r+1}{2^k},$$

take from the table the indices that correspond to r and r+1 and then perform binary search in order to find the exact non-zero element that is needed. We found that some care should be taken in the exact implementation of this procedure in software in order to achieve good scalability, but even when the mean number of non-zero elements per row or column is not high, we still observed significant speedup

versus the nave implementation. One important advantage of the tabular method is that the result is the same, so no additional theoretical justification is needed for using Quasi-Monte Carlo methods. This would not be the case when using something like the so-called Robinhood method (see, e.g., [31]).

4. Parallel implementation

The parallel implementation of the Monte Carlo or Quasi-Monte Carlo methods for matrix computations is usually based on splitting the Markov chains among processors. In the case when multiple right-hand-sides are present, for example in the matrix inversion problem, it is also natural to divide these right-hand-sides among processors. When accelerators are used one should be aware that the total available memory is usually less than what regular CPU-based servers offer. For example, the Xeon Phi 7120P coprocessor has just 16 GB of RAM. That is why it is desirable to combine OpenMP and MPI in the parallel implementation. We chose to split along the right-hand-sides between the different MPI processes and then along the Markov chains (also called trajectories) between the OpenMP threads. Because the number of Markov chains is given in advance, the blocking parallelisation approach for the Quasi-Monte Carlo algorithm can be used and thus our optimisations aimed at saving memory for the OpenMP parallelisation approach come into play. One consideration that is specific to matrix computations is that since the length of the Markov chains may vary, we have a problem to select the appropriate dimension for the Quasi-Monte Carlo sequence. We point out that our generators for Xeon Phi are optimised for dimensions that are multiples of 16. We can select a dimension based on theoretical estimate for the length of the Markov chain, if we know for example an estimate for the maximal eigenvalue of the matrix. In any case, once we chose certain dimension d, if we arrive at a situation where the Markov chain is larger than d, we can always use pseudo-random numbers for the remaining dimensions. Theoretically this is justified by the consideration that the last dimensions have lesser contribution in the overall result than the first dimensions. Nevertheless, one may simply re-do the pre-processing part of the generation of the pseudo-random numbers with higher dimension, using a running maximum, thus obtaining a pure Quasi-Monte Carlo algorithm. Our generators for the Sobol's sequence have faster pre-processing than our generators for the Halton sequence, so this approach will have lesser impact on overall running time for the Sobol's sequences. In the section with numerical results one can see how such a method works in practice.

Saving memory bandwidth while implementing Quasi-Monte Carlo algorithms. Once we optimised the memory requirements of our generation routines, we consider the memory bandwidth that they use. In general the speed of access to memory, measured with bandwidth and latency, improves at much slower rate compared to the speed of processing. That is why many computations are actually memory-bound. There are different ways to improve the execution through

optimisation of memory access. One can decrease the total amount of data being transferred or tune the patterns of access in order to improve the use of the various caches. For the Sobol sequence, there is one low-hanging fruit to be had in this direction. It is the observation that if we are generating consecutive terms of the sequence, half of the time the direction number being used is actually the same over all dimensions – it corresponds to changing the most significant binary digit after the binary point. Thus we do not need to load this number from memory. The resulting improvement is substantial in any kind of benchmark and may be even more important in real usage, since space in the caches is saved. Taking this approach one step further, we can make use of the fact that the matrices of binary numbers in the Sobol sequences are triangular. This means that for the first 8 positions 8 bits or one byte is enough to hold all the necessary information. Since we generate in double precision we usually need to load 64 bits or 8 bytes. Thus the savings in memory bandwidth are substantial, when we compress the corresponding "twisted direction numbers". The expansion happens with vector operations, by shifting appropriately and adding the omitted zeroes. Unfortunately, the Xeon Phi seems not to be efficient in such kinds of integer operations and thus this approach does not outperform in benchmarks. However, our benchmarks do not strain the use of caches and therefore cannot capture the advantage of this approach. On the CPUs similar approach has been winning in previous tests. Since the special handling of the first direction number had clear advantage, we leave the choice of using the compression for the next 7 direction numbers to the user. The generation codes are provided under the GNU Public license and are available at **http://parallel.bas.bg/~emanouil/sequences/micmemory.tgz**

5. Numerical tests

The justification for the use of Quasi-Monte Carlo methods in linear algebra problems is mainly in the hope of achieving better precision than regular Monte Carlo. In Fig. 3 we show the mean-squared error that was achieved when using the Sobol and (modified) Halton sequences to solve a linear system with 100 right-hand-sides with a matrix A that is diagonally dominant and sparse. We can see that using the Halton sequences one can significantly outperform the usual Monte Carlo method. The Sobol sequences show actually worse results than Monte Carlo, which shows that the selection of the low-discrepancy sequence is important for the QMC application. Next, in the Table 2 we can see a comparison of the computing times and parallel speedup in different settings. One can see that for this kind of linear algebra problems the differences in generation speed among the quasi- and pseudo-random sequences are noticeable, but not critical, and thus it is justified to use the sequences that yield the best accuracy (in this case, the modified Halton sequences).

Table 2. Comparison of execution times and speedup using different number of threads during the computation

Cards	Threads	Sobol		MT2203		Halton	
		Time, s	Speedup	Time, s	Speedup	Time, s	Speedup
1	1	109.05		107.61		132.12	
	61	3.21		3.23		3.58	
	122	2.97	1.08	2.99	1.08	3.24	1.10
	244	3.71	0.87	3.69	0.87	3.83	0.93
2	1	52.12		56.33		51.62	
	61	1.92	1.67	2.14	1.51	2.11	1.69
	122	1.63	1.97	1.75	1.85	1.93	1.85
	244	8.32	0.38	8.89	0.36	8.35	0.43
8	1	14.03		13.72		16.11	
	61	0.55	5.83	0.54	5.98	0.63	5.68
	122	0.69	4.65	0.62	5.21	0.67	5.34
	244	1.03	3.12	0.93	3.47	1.00	3.58
16	1	7.24		7.53		9.00	
	61	0.38	8.45	0.35	9.23	0.42	8.53
	122	0.49	6.55	0.44	7.34	0.53	6.75
	244	0.68	4.72	0.80	4.04	0.84	4.26

We notice that the best choice of hyperthreading seems to be to use either 61 or 122 logical threads, which means no hyperthreading or $2x$ hyperthreading, instead of the maximum feasible 244 threads, corresponding to hyperthreading with a factor of four. In this kind of problem the gain from hyperthreading, when it happens, seems to be limited, which may be due to the fact that the algorithms use lots of memory bandwidth. The increase in the number of computing devices and consequently, the number of MPI processes, leads to better results when hyperthreading is not used. In some cases the use of 4x hyperthreading significantly degrades the performance. When comparing the speedup with different number of devices, when the basis is the case of 61 threads on one card, we see that the speedup is acceptable, but far from the perfect linear speedup. However, the actual computing times are in the range of a few seconds, which is rather small and so some pre-processing steps that are not fully parallelized, have large impact on the results. In the next figure one can see the features of some of the test matrices that we have used.

When the algorithm involves constructing a preconditioner, we observe a breakdown of the total used wall clock time between the various stages, which shows that actually the slowest part is the broadcasting, which saturates the intra-node bandwidth. Thus when considering execution in multi-node environment one has to concentrate on decreasing the broadcasting time. It is possible to use some tuning parameters of the MPI library in order to obtain maximum performance on the particular hardware setup. It is also important to condense the matrix before broadcasting. However, this should be done via simple algorithm, so that this operation does not impact adversely the total execution time.

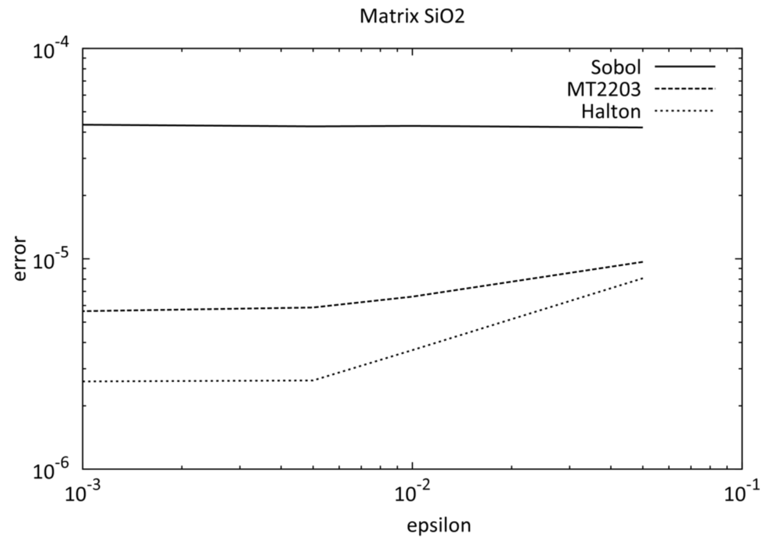

Fig. 3. Mean squared error for three algorithms: Monte Carlo (using MT), Quasi-Monte Carlo with Sobol sequence and Quasi-Monte Carlo with Halton sequence

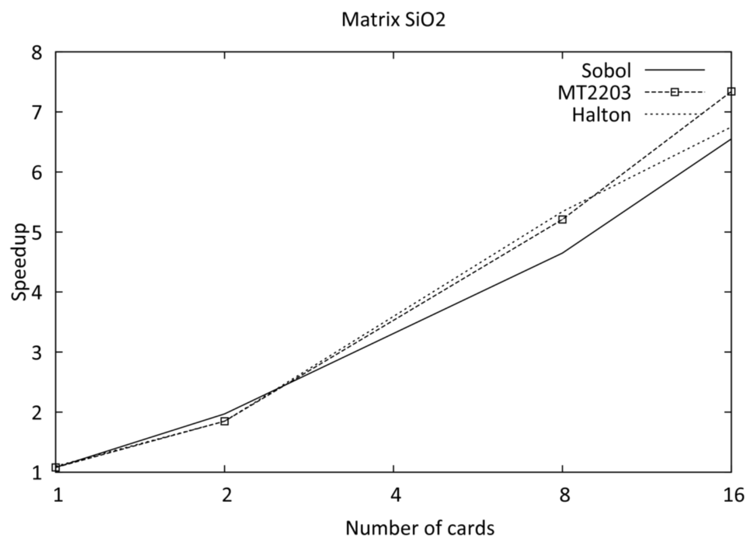

Fig. 4. Speedup when using 122 threads on varying number of cards, compared with 61 threads on one card

6. Conclusions and future work

The Monte Carlo algorithms for solving linear algebra problems have established place when the problem at hand has certain characteristics. The Quasi-Monte Carlo algorithms intend to improve on them by increasing the accuracy while maintaining similar speed of execution. We have seen how the use of some families of

quasirandom sequences, like the modified Halton sequences, can achieve these results, without changing significantly the program code or the flow of the algorithm. The selection and implementation of the tabular method have taken into account specific properties of the quasirandom sequences, enabling us to actually achieve improved performance. It is possible that other methods for sampling discrete distributions may also be adapted for quasirandom numbers, which will be subject to further research.

The achieved scalability using Intel Xeon Phi is acceptable for practical use. We believe it is possible to further optimise the code to take into account the vector processing capabilities of these devices. We intend to continue in this direction of research, using either compiler directives or intrinsic functions.

***Acknowledgments*:** This work was partially supported by the European Commission under H2020 Project VI-SEEM (Contract Number 675121) by the National Science Fund of Bulgaria under Grant DFNI-I02/8.

References

1. Alexandrov, V. N., O. A. Esquivel-Flores. Towards Monte Carlo Preconditioning Approach and Hybrid Monte Carlo Algorithms for Matrix Computations. – Computers & Mathematics with Applications, Vol. **70**, 2015, Issue 11, pp. 2709-2718. ISSN 0898-1221. **https://doi.org/10.1016/j.camwa.2015.08.035**
2. Alexandrov, V., A. Karaivanova. Parallel Monte Carlo Algorithms for Sparse SLAE Using MPI. – In: J. Dongarra, E. Luque, T. Margalef, Eds. LNCS. Vol. **1697**. Springer, 1999, pp. 283-290.
3. Alexandrov, V., O. Esquivel-Flores, S. Ivanovska, A. Karaivanova. On the Preconditioned Quasi-Monte Carlo Algorithm for Matrix Computations. – In: LNCS. Vol. **9374**. Springer, 2015, pp. 163-171.
4. Atanassov, E. I. On the Discrepancy of the Halton Sequences. – Mathematica Balkanica, Vol. **18**, 2004, Fasc. 1-2, pp. 15-32.
5. Atanassov, E. I. A New Efficient Algorithm for Generating the Scrambled Sobol' Sequence. Numerical Methods and Applications. – In: LNCS. Vol. **2542**. Springer-Verlag. 2003, pp. 83-90.
6. Atanassov, E., T. Gurov, S. Ivanovska, A. Karaivanova. Parallel Monte Carlo on Intel MIC Architecture. – Procedia Computer Science, Vol. **108**, 2017, pp. 1803-1810. **https://doi.org/10.1016/j.procs.2017.05.149**
7. Atanassov, E., T. Gurov, A. Karaivanova, S. Ivanovska, M. Durchova, D. Dimitrov. On the Parallelization Approaches for Intel MIC Architecture. – In: AIP Conf. Proc. 1773, 070001, 2016. **http://dx.doi.org/10.1063/1.4964983**
8. Boisvert, R. F. et al. Matrix Market: A Web Resource for Test Matrix Collections. – In: R. F. Boisvert, Ed. Quality of Numerical Software. IFIP Advances in Information and Communication Technology. Boston, Springer, MA, 1997, pp. 125-137.
9. Caflisch, R. Monte Carlo and Quasi-Monte Carlo Methods. – Acta Numerica, Vol. **7**, 1998, pp. 1-49.
10. Curtiss, J. H. Monte Carlo Methods for the Iteration of Linear Operators. – J. of Math. Physics, Vol. **32**, 1954, pp. 209-232.
11. Danilov, D., S. Ermakov, J. H. Halton. Asymptotic Complexity of Monte Carlo Methods for Solving Linear Systems. – J. of Stat. Planning and Inference, Vol. **85**, 2000, pp. 5-18.
12. Davis, T. A., Y. Hu. SuiteSparse Matrix Collection. – ACM Transactions on Mathematical Software (TOMS), Vol. **38**, 2011, No 1, p. 1. **www.cise.ufl.edu/research/sparse/matrices/**

13. D i m o v, I., V. A l e x a n d r o v, A. K a r a i v a n o v a. Resolvent Monte Carlo Methods for Linear Algebra Problems. – Math. and Comp. in Simulations, Vol. **55**, 2001, pp. 25-36.
14. D i m o v, I., A. K a r a i v a n o v a. Parallel Computations of Eigenvalues Based on a Monte Carlo Approach. – Monte Carlo Methods and Applications, Vol. **4**, 1998, No 1, pp. 33-52.
15. F a t h i, B., B. L i u, V. A l e x a n d r o v. Mixed Monte Carlo Parallel Algorithms for Matrix Computation. – In: LNCS. Vol. **2330**. Springer, 2002, pp. 609-618.
16. F o r s y t h e, G., R. L e i b l e r. Matrix Inversion by a Monte Carlo Method. – Math. Tables and other Aids to Computation, Vol. **4**, 1950, pp. 127-147.
17. G r o t e, M., M. H a g e m a n n. SPAI: SParse Approximate Inverse Preconditioner, Spaidoc. – Pdf paper in the SPAI, Vol. **3**, 2006, p. 1.
18. H a m m e r s l e y, J., D. H a n d s c o m b. Monte Carlo Methods. New York, London, Sydney, John Wiley & Sons, 1964.
19. H a l t o n, J. H. Sequential Monte Carlo. – Proceedings of the Cambridge Philosophical Society, Vol. **58**, Part 1, 1962, pp. 57-78.
20. H a l t o n, J. H. Sequential Monte Carlo Techniques for the Solution of Linear System. – IAM J. of Sci. Comp., Vol. **9**, 1994, pp. 213-257.
21. H u c k l e, T., et al. An Efficient Parallel Implementation of the MSPAI Preconditioner. – Par. Computing, Vol. **36**, 2010, No 5-6, pp. 273-284.
22. K a r a i v a n o v a, A. Quasi-Monte Carlo Methods for Some Linear Algebra Problems. Convergence and Complexity. – Serdica J. of Comp., Vol. **4**, 2010, pp. 58-72.
23. K r o e s e, D. P., T. T a i m r e, Z. I. B o t e v. Handbook of Monte Carlo Methods. John Wiley & Sons, 2011.
24. M a s c a g n i, M., A. K a r a i v a n o v a. Matrix Computations Using Quasirandom Sequences. – In: LNCS. Vol. **1988**. Springer, 2001, pp. 552-559.
25. M a s c a g n i, M., A. K a r a i v a n o v a. A Parallel Quasi-MCM for Computing Extremal Eigenvalues. – In: MCQMCMs 2000. Springer, 2002, pp. 369-380.
26. S o b o l, I. Monte Carlo Numerical Methods. Moscow, Nauka, 1973 (in Russian).
27. S t o y k o v, S., E. A t a n a s s o v, S. M a r g e n o v. Efficient Sparse Matrix-Matrix Multiplication for Computing Periodic Responses by Shooting Method on Intel Xeon Phi. – In: AIP Conference Proceedings, 1773, 110012, 2016.
http://dx.doi.org/10.1063/1.4965016
28. S t r a ß b u r g, J., V. N. A l e x a n d r o v. Enhancing Monte Carlo Preconditioning Methods for Matrix Computations. – In: Proc. of ICCS 2014, pp. 1580-1589.
29. V a j a r g a h, B. F. A New Algorithm with Maximal Rate Convergence to Obtain Inverse Matrix. –Applied Mathematics and Computation, Vol. **191**, 2007, No 1, pp. 280-286.
http://dx.doi.org/10.1016/j.amc.2007.02.085
30. W a s o w, W. A Note on the Inversion of Matrices by Random Walks. – Math. Tables and other Aids to Computation, Vol. **6**, 1952, pp. 78-81.
31. W e s t l a k e, J. A Handbook of Numerical Matrix Inversion and Solution of Linear Equations. New York, J. Wiley & Sons, 1968.
32. Y i m u, J., K. Z i z h u o, P. Q. Y u, S. Y a n p e n g, K. J i a n g b a n g, H. W e i. A Cloud Computing Service Architecture of a Parallel Algorithm Oriented to Scientific Computing with CUDA and Monte Carlo. – Cybernetics and Information Technologies, Vol. **13**, 2013, Special Issue, pp. 153-166.
33. Intel® Xeon Phi™ Coprocessor Instruction Set Architecture Reference Manual.
https://software.intel.com/sites/default/files/forum/278102/327364001en.pdf