Scalable Computing: Practice and Experience
Volume 11, Number 1, pp. 73–80. http://www.scpe.org
ISSN 1895-1767


# PARALLEL QUASIRANDOM APPLICATIONS ON HETEROGENEOUS GRID

A. KARAIVANOVA, E. ATANASSOV, T. GUROV, S. IVANOVSKA, M. DURCHOVA*

**Abstract.** In this paper we present error and performance analysis of quasi-Monte Carlo algorithms for solving multidimensional integrals (up to 100 dimensions) on the grid using MPI. We take into account the fact that the Grid is a potentially heterogeneous computing environment, where the user does not know the specifics of the target architecture. Therefore parallel algorithms should be able to adapt to this heterogeneity, providing automated load-balancing. Monte Carlo algorithms can be tailored to such environments, provided parallel pseudorandom number generators are available. The use of quasi-Monte Carlo algorithms poses more difficulties. In both cases the efficient implementation of the algorithms depends on the functionality of the corresponding packages for generating pseudorandom or quasirandom numbers. We propose efficient parallel implementation of the Sobol sequence for a grid environment and we demonstrate numerical experiments on a heterogeneous grid. To achieve high parallel efficiency we use a newly developed special grid service called *Job Track Service* which provides efficient management of available computing resources through reservations.

**1. Introduction.** Quasi-Monte Carlo methods and algorithms are proved to be efficient in many areas ranging from physics to economy. They are based on quasirandom (also called low discrepancy) sequences while Monte Carlo methods are based on pseudorandom numbers [3]. As a rule the computational schemes of the quasi-Monte Carlo methods are simple and convenient for use in applied computing. There has been renewed interest in quasi-Monte Carlo in recent times, the primary reason for this is that quasi-Monte Carlo methods usually give much better rate of convergence than Monte Carlo ones, and, in the same time, preserve the good parallel properties of Monte Carlo applications.

Monte Carlo and quasi-Monte Carlo are often used for solving computationally intensive problems. Studying their performance on different parallel and distributed computer systems is an important practical problem. For example, problems with dimension 100 and more are typical for financial mathematics. For such problems it is natural to use parallel and Grid systems. Nowadays, from the other hand, the computational Grid proved to be a very efficient computing model [12]. The Grid aims ultimately to turn the global network of computers into one vast computational resource [4]. Grid computing in general is a special type of parallel computing. Technically Grid coordinates resources which are not a subject to central administrative control and utilizes general-purpose protocols. Another distinction is that a Grid could in principle have access to parallel computers, clusters, farms, local Grids, even Internet computing solutions, and would choose the appropriate tool for a given calculation. In this sense, the Grid is the most generalized form of distributed computing.

One major advantage of Monte Carlo methods is that they are usually very easy to be parallelized. This is, in principal, also true of quasi-Monte Carlo methods. However, the successful parallel implementation of a quasi-Monte Carlo application depends crucially on various quality aspects of the parallel quasirandom sequences used [5, 6]. Much of the recent work on parallelizing quasi-Monte Carlo methods has been aimed at splitting a quasirandom sequence into many subsequences which are then used independently on various parallel processes, for example in [1, 7]. This method works well for the parallelization of pseudorandom numbers, but due to the nature of quality in quasirandom numbers, this technique has some difficulties. Our algorithms are based on scrambling which is suitable for heterogeneous computing environments.

In the present paper we propose and study parallel Sobol sequence application in heterogeneous grid environment. Sobol sequence is one of the most popular quasirandom sequences. We first present the algorithm for fast generation of scrambled Sobol sequence, parallelization techniques, and some numerical results for solving multidimensional integrals which show the quality of our generation. In the next section we present grid performance results for a quasi-Monte Carlo method intended for heavy computations (dimension 100). Our tests proved the applicability of the presented scrambled Sobol sequence in grid environment. Moreover, the MPI code is executed simultaneously on two different grid clusters (one with Mirynet interconnect, the other with conventional interconnect) and high parallel efficiency has been achieved. For the purpose of efficient implementation of MPI applications a special grid service called *Job Track Service* (JTS) has been developed and its initial version has been tested here.

*IPP-BAS, Acad. G. Bonchev St., bl. 25A, Sofia 1113, Bulgaria, E-mails: anet, emanouil, gurov, sofia, mabs@parallel.bas.bg

**2. Monte Carlo and Quasi-Monte Carlo Integration.** Consider an integral on the unit cube $I^s = [0,1]^s$ in $s$ dimensions,

$$I[f] = \int_{I^s} f(x)dx, \tag{2.1}$$

of a Lebesgue integrable function $f(x)$, and note that this integral can be expressed as the expectation of the function $f$, $I[f] = E[f(x)]$, where $x$ is a uniformly distributed vector in the unit cube.

Consider the following approximation of the integral (2.1):

$$I_N[f] = \frac{1}{N}\sum_{n=1}^{N} f(x_n). \tag{2.2}$$

If $\{x_n\}$ is a sequence sampled from uniform distribution, equation (2.2) is called (crude) Monte Carlo quadrature formula. The integration error, defined as

$$\epsilon_N[f] = |I[f] - I_N[f]|, \tag{2.3}$$

has a standard normal distribution, with expectation

$$E[\epsilon_N[f]] = \sqrt{Var(f)}N^{-1/2}. \tag{2.4}$$

An exact upper bound for (2.2) is given by Koksma-Hlawka inequality,

$$\epsilon_N[f] \leq V[f]D_N^*, \tag{2.5}$$

in which V[f] is the variation of $f$ in the Hardy-Krause sense and $D_N^*$ is the star discrepancy of the sequence $\{x_n\}$, [3]. Equation (2.5) is valid for any function with bounded variation and any choice of sequence [3], however the best results are obtained with low discrepancy (also called quasirandom) sequences for which

$$D_N^* \leq c(\log N)^k N^{-1},$$

where $c$ and $k$ are constants independent of $N$ but dependent on $s$.

In this paper we use scrambled Sobol sequence. The purpose of using scrambled sequences in quasi-Monte Carlo methods is twofold. Primarily, it provides a practical method to obtain error estimates for QMC based on treating each scrambled sequence as a different and independent random sample from a family of randomly scrambled quasirandom numbers [1, 11]. Thus, randomized QMC overcomes the main disadvantage of QMC while maintaining the favorable convergence rate of QMC. Secondarily, scrambling gives us a simple and unified way to generate quasirandom numbers for parallel, distributed, and Grid-based computational environments [4, 5, 6].

The test functions for numerical integration in this paper are:

$$F_1 = \prod_{i=1}^{s}\left({x_i}^3 + \frac{3}{4}\right),$$

$$F_2 = \prod_{i=1}^{s}|4x_i - 2|.$$

The first of test function is described in [8]. The second is known as Roos and Arnold's example and it is suggested as test function in [9].

**3. Sobol sequences.** Sobol sequence proved to be one of the most efficient sequences for quasi-Monte Carlo integration, [2, 3, 10]. The original sequence is not suitable for grid implementation, [5, 6]. Here we present a fast algorithm for generation of a scrambled Sobol sequence suitable for parallel and grid implementations. We use the Definition below, which covers most digital $(t, m, s)$-nets in base 2. The Sobol sequence is a $(t, s)$-sequence in base 2 and is a particular case of this definition.

**Definition.** *Let $A_1, \ldots, A_s$ be infinite matrices $A_k = \{a_{ij}^{(k)}\}$, $i, j = 0, 1, \ldots$, with $a_{ij}^{(k)} \in \{0, 1\}$, such that $a_{ii}^{(k)} = 1$ for all $i$ and $k$, $a_{ij}^{(k)} = 0$ if $i < j$. The $\tau^{(1)}, \ldots, \tau^{(s)}$ are sequences of permutations of the set $\{0, 1\}$. Each non-negative integer $n$ may be represented in the binary number system as*

$$n = \sum_{j-0}^{r} b_i 2^i.$$

*Then the $n$-th term of the low-discrepancy sequence $\sigma$ is defined by*

$$x_n^{(k)} = \sum_{j=0}^{r} 2^{-j-1} \tau_j^{(k)} (\oplus_{i=0}^{j} b_i a_{ij}^{(k)}),$$

*where by "$\oplus$" we denote the operation "bit-wise addition modulo 2".*

Next lemma explains how we generate consecutive terms of the sequence.

**Lemma.** *Let $\sigma$ be a sequence or net satisfying the above Definition, and let the non-negative integers $n, p, m$ be given. Suppose that we desire the first $p$ binary digits of elements in $\sigma$ with indices of the form $2^m j + n < 2^p$. This implicitly defines a set of compatible $j's$. Thus the only numbers we need to compute are*

$$y_j^{(k)} = \lfloor 2^p x_{2^m j+n}^{(k)} \rfloor.$$

*The integers $\{v_r^{(k)}\}_{r=0}^{\infty}$, which we call "twisted direction numbers", are defined by*

$$v_r^{(k)} = \sum_{t=0}^{p-1} 2^{p-1-t} \oplus_{j=m}^{p-1} a_{tj}^{(k)}.$$

*Suppose that the largest power-of-two that divides $2^m(j+1) + n$ is $l$, i. e. $2^m(j+1) + n = 2^i(2K+1)$. Then the following equality holds*

$$y_{j+1}^{(k)} = y_j^{(k)} \oplus v_i^{(k)}.$$

The main algorithm for generating scrambled Sobol sequence is described in [1]. Here we study the quality and the applicability of the generated sequence in grid environment. The algorithm allows consecutive terms of the scrambled sequence to be obtained with essentially only two operations per coordinate: one floating point addition and one bit-wise xor operation (this omits operations that are needed only once per tuple). This scrambling is achieved at no additional computational cost over that of unscrambled generation as it is accomplished totally in the initialization. In addition, the terms of the sequence are obtained in their normal order, without the usual permutation introduced by Gray code ordering used to minimize the cost of computing the next Sobol element. This algorithm is relatively simple and very suitable for parallel and grid implementation.

The mathematical explanations can be found in [1] and [2], here we present the algorithm in pseudo code.

- Input initial data:
  - if the precision is single, set the number of bits $b$ to 32, and the maximal power of two $p$ to 23, otherwise set $b$ to 64 and $p$ to 52;
  - dimension $s$;
  - direction vectors $\{a_{ij}\}, i = 0, p, j = 1, \ldots, s$ representing the matrices $A_1, \ldots, A_d$ (always $a_{ij} < 2^{i+1}$);
  - scrambling terms $d_1, \ldots, d_s$ - arbitrary integers less than $2^p$, if all of them are equal to zero, then no scrambling is used;
  - index of the first term to be generated - $n$;
  - scaling factor $m$, so the program should generate elements with indices $2^m j + n$, $j = 0, 1, \ldots$.
- Allocate memory for $s \star l$ $b$-bit integers (or floating point numbers in the respective precision) $y_1, \ldots, y_s$.
- Preprocessing: calculate the twisted direction numbers $v_{ij}$, $i = 0, \ldots, p-1$, $j = 0, \ldots, s$:
  - for all $j$ from 1 to $s$ do
  - for $i = 0$ to $p - 1$ do

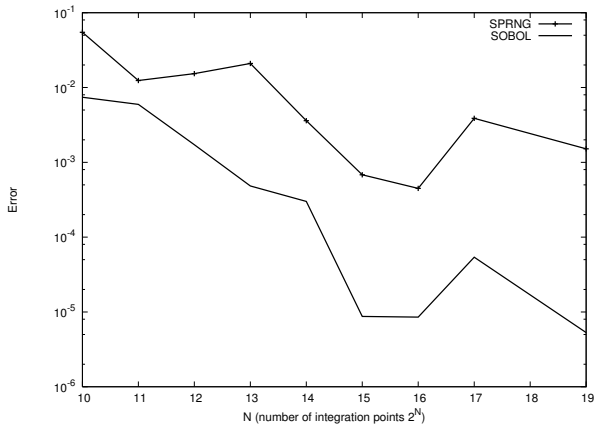


FIG. 3.1. *Accuracy of numerical integration of* **Test function** $F_1$ *with sample size* $N$ *and dimension 10 using pseudorandom numbers and Sobol sequence.*

  - if $i = 0$, then $v_{ij} = a_{ij}2^{p-m}$,
    else $v_{ij} = v_{i-1j}\mathbf{xor}(a_{i+m,j} \star (2^{p-i-m}))$;
- Calculate the coordinates of the $n^{th}$ term of the Sobol sequence (with the scrambling applied) using any known algorithm (this operation is performed only once). Add 1 to all of them and store the results as floating point numbers in the respective precision in the array $y$.
- Set the counter $N$ to the integer part of $\dfrac{n}{2^m}$.
- Generate the next point of the sequence:
  - When a new point is required, the user supplies a buffer $x$ with enough space to hold the result.
  - The array $y$ is considered as holding floating point numbers in the respective precision, and the result of subtracting 1 from all of them is placed in the array $x$.
  - Add 1 to the counter $N$.
  - Determine the first nonzero binary digit $k$ of $N$ so that $N = (2M + 1)2^k$ (on the average this is achieved in 2 iterations).
  - Consider the array $y$ as an array of $b$-bit integers and updated it by using the $k^{th}$ row of twisted direction numbers:
    for $i = 1$ to $d$ do $y_i = y_i\mathbf{xor}v_{ki}$.
  - Return the control to the user. When a new point is needed, go to beginning of this paragraph (Generate the next point of the sequence).

Numerical results for approximate calculation of 10 and 20 dimensional integrals with Monte Carlo and quasi-Monte Carlo using scrambled Sobol sequence are shown on Figure 3.1 and Figure 3.2 (test function $F_1$) and Table 3.1 (test function $F_2$). Let us mention that usually dimension 20 is considered as too high for quasi-Monte Carlo treatment. Here we see that the quasi-Monte Carlo algorithm with scrambled Sobol sequence (Sobol in the Table3.1) gives better results than Monte Carlo (SPRNG available on [14] in the Table 3.1). The good properties of the sequence motivated us for using this generator in our grid applications.

**Parallelization.**

There are three basic ways to parallelize quasirandom number sequences:

- *Leap-frog*—the sequence is partitioned in turn among the processors like a deck of cards dealt to card players.
- *Sequence splitting or blocking*—the sequence is partitioned by splitting it into non-overlapping contiguous subsections.
- *Independent sequences*—each processor has its own independent sequence.

The first and second schemes produce numbers from a single quasirandom sequence. The third scheme needs a family of quasirandom sequences. Scrambling techniques can generate such a stochastic family of quasirandom sequences from one original quasirandom sequence. Numerical calculations presented in this paper are done using scrambling and blocking. In this way we can exactly repeat the results for comparisons.

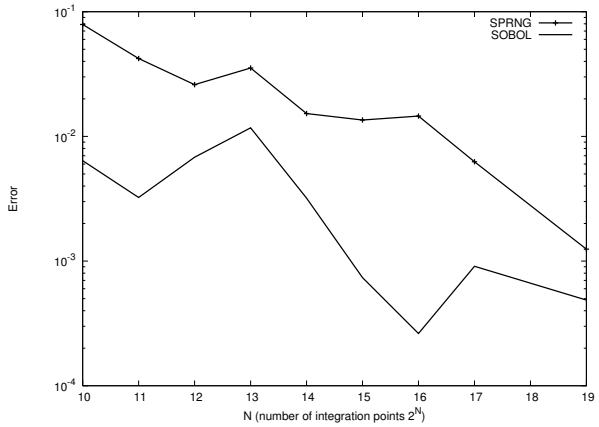


FIG. 3.2. *Accuracy of numerical integration of* **Test function** $F_1$ *with sample size N and dimension 20 using pseudorandom numbers and Sobol sequence.*

TABLE 3.1
*Accuracy of numerical integration of* **Test function** $F_2$ *with sample size N when using pseudorandom numbers and Sobol sequence.*

| $N$ | dim | SPRNG | Sobol |
|---|---|---|---|
| $2^{10}$ | 10 | 7.033e-02 | 1.220e-02 |
| | 20 | 3.164e-01 | 2.114e-01 |
| $2^{11}$ | 10 | 2.622e-02 | 2.504e-02 |
| | 20 | 2.851e-02 | 1.331e-02 |
| $2^{12}$ | 10 | 1.106e-01 | 2.225e-02 |
| | 20 | 9.003e-02 | 4.579e-02 |
| $2^{13}$ | 10 | 4.254e-02 | 3.722e-02 |
| | 20 | 1.454e-01 | 1.403e-01 |
| $2^{14}$ | 10 | 4.777e-02 | 1.667e-02 |
| | 20 | 1.064e-02 | 8.385e-02 |
| $2^{15}$ | 10 | 3.482e-02 | 1.456e-02 |
| | 20 | 3.751e-02 | 1.514e-01 |
| $2^{16}$ | 10 | 2.622e-02 | 2.919e-03 |
| | 20 | 1.528e-02 | 3.104e-02 |
| $2^{17}$ | 10 | 2.098e-02 | 2.059e-03 |
| | 20 | 2.966e-02 | 8.379e-03 |
| $2^{19}$ | 10 | 2.024e-03 | 2.475e-05 |
| | 20 | 1.190e-02 | 1.716e-02 |

**4. Quasi-Monte Carlo Computations on the Grid.** We take into account the fact that the Grid is a potentially heterogeneous computing environment, where the user does not know the specifics of the target architecture. Therefore parallel algorithms should be able to adapt to this heterogeneity, providing automated load-balancing. Monte Carlo algorithms can be tailored to such environments, provided parallel pseudorandom number generators are available.
The use of quasi-Monte Carlo algorithms poses more difficulties. In both cases the efficient implementation of the algorithms depends on the functionality of the corresponding packages for generating pseudorandom or quasirandom numbers.

**4.1. Grid-enabled Parallel Implementation of the Algorithm for Numerical Integration.** Our tests are implemented on the two main Grid clusters in IPP-BAS which are included in the Pan-European (EGEE) and regional (SEE-GRID) grid infrastructures and which have completely different characteristics.

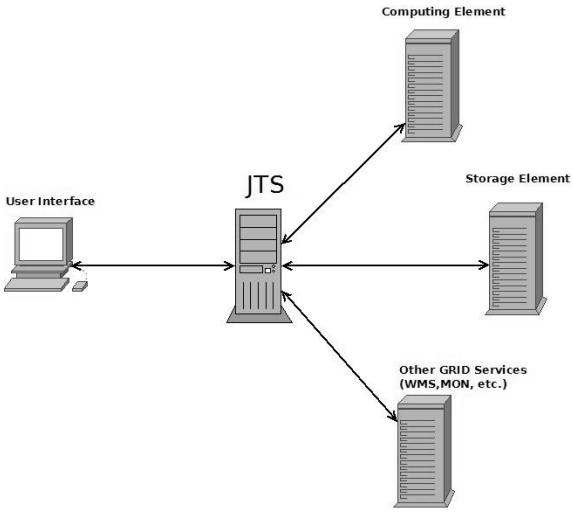


FIG. 4.1. *JTS working scheme.*

EGEE (www.eu-egee.org) grid infrastructure serves a wide range of European scientific communities and is currently the largest, most widely used multidisciplinary grid infrastructure in the world [12]. The EGEE grid uses gLite (www.glite.org), an open source middleware distribution that combines components developed in various related projectsin particular Condor and Globusvia the Virtual Data Toolkit.

The parameters of the grid clusters used in our experiments are: The BG03-NGCC grid cluster has 25 worker nodes, 2 x Intel Xeon E5430 2.66 GHz Quad Core CPU (total 200 Cores, $>$ 400 kSI2000). Each node has 16 GB RAM. The BG04-ACAD has 40 worker nodes with 2 CPU Opteron 2,4 GHz (total 80 cores $>$ 120 KSI2000), 4GB RAM for each node, and low-latency Myrinet interconnection for MPI jobs as in [13]. Here we present grid-enabled parallel implementation of the standard Monte Carlo and quasi-Monte Carlo methods for numerical integration. In our grid tests we use the function $F_1$ with dimension $s = 100$.

The parallelization is based on the master-slave paradigm, with some ramifications. We divide the work into chunks, corresponding to the sub-domains, which are requested from the master process by the slave processes. In order to increase the efficiency, the master also performs computations, while waiting for communication requests. Thus we achieve overlap of computations and communications, and we do not lose the possible output of the master process. When using low-discrepancy sequences, we take care in both master and slave processes to fast-forward the generator exactly to the point that is needed. The scrambling that is provided by the generators enables a posteriori estimation of the error in the quasi-Monte Carlo case.

**4.2. Job Track Service.** In order to run MPI application simultaneously on different grid clusters we use newly developed service called Job Track Service (JTS) which allows CPU reservation for parallel jobs, and collecting and analysis of the results of these parallel jobs. JTS (see Fig. 4.1) implements messaging system to allow Quality of Service (QoS) for jobs, gathering and analysis of performance data and provision of application specific info from central point.

Some details about JTS:

- The core of the JTS is a JTS server. In this initial version there is only one instance of this server, located at IPP.
- The JTS server accepts connections via AMQP (Advanced Message Queuing Protocol, www.amqp.org), XMPP (Extensible Messaging and Presence Protocol, www.xmpp.org), and SOAP.
- The grid clients are running on the Grid Computing Elements. They connect with the JTS server using AMQP and send/receive messages, related to jobs.
- The users can send additional performance information or can manipulate jobs in a desired way. For instance, they can tag jobs as higher priority and they can direct jobs to VO reservations, available to their application.

The JTS provides efficient management of available computing resources through reservations. Systems of this kind can interoperate and probably will become the standard 2-3 years from now. The potential for

TABLE 4.1
*Parallel efficiency measurements for $N = 10^9$ points - $T_1$ is the theoretically optimal time and $T_2$ is the measured time.*

| | $T_1$ | $T_2$ | Efficiency |
|---|---|---|---|
| SPRNG | 117.10 | 129.87 | 90.17% |
| Sobol | 14.37832 | 15.34 | 93% |

TABLE 4.2
*Time for* **test function** $F_1$ *with* $10^8$ *points.*

| | 1 CPU (BG03) | 1 CPU (BG04) |
|---|---|---|
| SPRNG | 785.64 | 533.90 |
| Sobol | 65.79 | 95.65 |

standards-based (AMQP, STOMP, HTTP, XMPP) integration of information from different services and easy presentation to end user/site admin/VO admin can enhance the utilization and experience while using the Grid infrastructure. This trend is visible in the Cloud computing world (see, e.g., ElasticFox, S3Fox).

In our computations we use the JTS in order to ensure simultaneous startup of the MPI jobs on the two chosen clusters.

**4.3. Numerical Experiments.** Our numerical tests are performed for 100-dimensional integrals which is usual dimension for many applications in financial mathematics. The presented results are from parallel implementation of our algorithm on *20 CPU cores on BG04-ACAD* and *32 CPU cores on BG03-NGCC* simultaneously. The start of these MPI jobs on both clusters has been achieved by calling the Job Track Service on these two clusters.

In Table 4.1. we compare the estimated time for running our algorithm on the Worker Nodes of the grid sites BG03-NGCC and BG04-ACAD in parallel, in case of perfect parallel efficiency, with the measured execution time. Here $T_1$ is theoretically optimal time, $T_2$ is measured time and the efficiency is the ratio $\frac{T_1}{T_2}$. We used the MPICH-G2 implementation of MPI, which is the most general approach for running parallel jobs on computational grids.

We obtained roughly the same accuracy with low-discrepancy sequences as with pseudorandom numbers, due to the high effective dimension. The CPU time in Table 4.2 of our implementations of the quasi-Monte Carlo versions was significantly smaller, due to the efficient generation algorithms. Quasi-Monte Carlo algorithms are more difficult to parallelize due to the nature of the sequences, but this can be done if the generators of the low-discrepancy sequences have the necessary functionality.

**5. Conclusion and future work.** The purpose of this paper is to present and study quality of a fast generator of scrambled Sobol sequence, and its grid implementation. The comparative study of Monte Carlo and quasi-Monte Carlo method using Sobol sequence for computing very high-dimensional integrals (s=100) show the advantages of our generator even in heterogeneous computing environment. In any case high dimensional problems are good candidates for execution in the Grid environment, although the use of quasi-Monte Carlo or true random number generators poses certain technical challenges, related to the parallelization of the computations or the repeatability of results.

As a future work we plane to use other scrambling methods for Sobol sequence and its usage in large problems.

**6. Acknowledgment.** This work has been supported by the National Science Fund of Bulgaria through grants DO02-146 and DO02-115.

REFERENCES

[1] E. ATANASSOV: A New Efficient Algorithm for Generating the Scrambled Sobol' Sequence, *Numerical Methods and Applications, Springer*, vol. 2542 ,p. 83, 2003.

[2] S. IVANOVSKA, E. ATANASSOV, A. KARAIVANOVA: A Superconvergent Monte Carlo Method for Multiple Integrals on the Grid, *LNCS, Springer-Verlag, Berlin-Heidelberg,* vol. 3516, p. 735, 2005.

[3] R. Caflisch: Monte Carlo and quasi-Monte Carlo methods, *Acta Numerica,* vol. 7, p. 1, 1998.
[4] J. Foster, C. Kesselmann: *The Grid: Blueprint for a New Computing Infrastructure,* Morgan Kaufmann, 1998.
[5] H. Chi and E. Jones: Generating Parallel Quasirandom Sequences by using Randomization, *Journal of distributed and parallel computing,* vol. 67(7), p. 876, 2007.
[6] H. Chi and M. Mascagni: Efficient Generation of Parallel Quasirandom Sequences via Scrambling, *Lecture Notes in Computer Science, Springer,* vol. 4487,p. 723, 2007.
[7] B. C. Bromley: Quasirandom Number Generation for Parallel Monte Carlo Algorithms, *Journal of Parallel Distributed Computing,* vol. 38(1), p. 101, 1996.
[8] W. Schmid, A. Uhl: Techniques for parallel quasi-Monte Carlo integration with digital sequences and associated problems, *Math. and Comp. in Sim.,* vol. 55, p. 249, 2001.
[9] A. Owen: The Dimension Distribution and Quadrature Test functions, *Statistica Sinicai,* vol. 13, p. 1, 2003.
[10] S. Chaudhary: *Acceleration of Monte Carlo Methods using Low Discrepancy Sequences,* Dissertation, University of California, Los Angeles, 2004.
[11] G. Okten and B. Tuffin and V. Burago: A central limit theorem and improved error bounds for a hybrid-Monte Carlo sequence with applications in computational Finance, *Journal of Complexity,* vol. 22(4), p. 453, 2006.
[12] I. Bird, B. Jones, K. Kee: The organization and management of grid infrastructures, *IEEE Computer*, vol.9, p. 36, 2009.
[13] MPI website, `http://www-unix.mcs.anl.gov/mpi/`
[14] SPRNG: Scalable Parallel Random Number Generator, `http://sprng.cs.fsu.edu/`

*Edited by:* Dana Petcu
*Received:* January 12, 2010
*Accepted:* January 30, 2010