# PARMONC - A Software Library for Massively Parallel Stochastic Simulation[⋆]

Mikhail Marchenko

The Institute of Computational Mathematics and Mathematical Geophysics SB RAS,
Prospekt Lavrentieva 6, 630090, Novosibirsk, Russia,
Novosibirsk State University,
Ul. Pirogova 2, 630090, Novosibirsk, Russia
Tel.: (383)330-77-21, Fax: (383) 330-87-83
mam@osmf.sscc.ru

**Abstract.** In this paper, the software library PARMONC that was developed for the massively parallel simulation by Monte Carlo method on supercomputers is presented. The "core" of the library is a well tested, fast and reliable long-period parallel random numbers generator. Routines from the PARMONC can be called in the user-supplied programs written in C, C++ or in FORTRAN without explicit usage of MPI instructions. Routines from the PARMONC automatically calculate sample means of interest and the corresponding computation errors. A computational load is automatically distributed among processors in an optimal way. The routines enable resuming the simulation that was previously performed and automatically take into account its results. The PARMONC is implemented on high-performance clusters of the Siberian Supercomputer Center.

**Keywords:** Monte Carlo method, distributed stochastic simulation, random number generator, parallel computation, supercomputers.

## 1 Introduction

It becomes a common point of view that probabilistic imitation models and Monte Carlo method (stochastic simulation) will be widely used for computer-aided simulation in the nearest future. There are a few reasons for such a prediction. First of all, the use of probabilistic models is an adequate way to simulate physical, chemical or biologic phenomena from "first principles". On the other hand, Monte Carlo methods, which realize probabilistic models, can be effectively parallelized in the form of distributed computing. Therefore, a progress in development and implementation of powerful supercomputers gives way to a wide use of Monte Carlo method as a principal instrument for the computer-aided simulation in many scientific areas (see, e.g., [1], [2]).

The main objective of this paper is to introduce PARMONC - a library of easy-to-use programs that was implemented on high-performance clusters of the

[⋆] This work was supported by the RFBR grants No 09-01-00639 and No 09-01-00035.

Siberian Supercomputer Center (`http://www2.sscc.ru`) and can also be used in other supercomputer centers [3], [4].

The development of the PARMONC (an acronym for PARallel MONte Carlo) is based on the library MONC that was implemented for a network of personal computers [5]. The MONC was intensively used in the Department of Stochastic Simulation in Physics of the Institute of Computational Mathematics and Mathematical Geophysics of the SB RAS in Novosibirsk for a wide area of applications. Also, the MONC was actively applied in the Laboratory of Probability-Theoretical Methods of the Omsk Branch of the Sobolev Institute of Mathematics of the SB RAS to solve various problems in the population biology.

The main objectives of the library development are as follows:

- creation of a software tool suitable for the massively parallel stochastic simulation for a wide range of applications,
- creation of an easy-to-use software framework to parallelize stochastic simulation to be applied without knowledge of MPI language.

There are a number of publications and internet resources dedicated to the parallel random numbers generation (see, e.g., [6], [7], [8]). In comparison to other parallel random number generators (RNGs), the one used for the PARMONC is fairly fast, reliable and has an extremely long period. Also, this parallel RNG is governed by a few parameters defined by the user. Using this generator, it appears possible to scale the stochastic simulation to sufficiently large (practically infinite) number of processors (see Sections 2.4 and 3.5).

A number of publications are devoted to the development of software packages for parallel computations with Monte Carlo method (see, e.g., [9], [10], [11]). Different hardware and software platforms are reported in these publications. In our opinion, the following features distinguish the PARMONC from other software tools and make it an easy-to-use instrument for specialists in the field of stochastic simulation:

- The only thing the user has to do in order to parallelize stochastic simulation is to write in C, C++ or in FORTRAN a sequential subroutine to simulate a single realization of a random object of interest and to pass its name to the PARMONC routines (see Sections 2.3, 3.2 and 4).
- In his/her sequential code, he/she uses a PARMONC function, which implements a parallel RNG, in a usual and convenient way (see Sections 2.3, 2.4 and 3.3).
- In the course of simulation, the PARMONC periodically calculates and saves in files the subtotal results of simulation and the corresponding computation errors (see Sections 2.2, 3.2 and 4).
- The PARMONC provides an easy-to-use technique to resume stochastic simulation after its termination with automatic averaging of the results of the previous simulation (see Sections 3.2 and 4).

## 2 Background

### 2.1 Estimators of Interest in Stochastic Simulation

Initially, Monte Carlo method (or stochastic simulation) was developed to solve problems of radiation transfer. In the last half of the 20-th century, the area of its applications became much wider. Theory of stochastic representations for solutions to equations of mathematical physics was developed. Using the theory, the corresponding numerical stochastic estimators were drawn up. Efficient algorithms of Monte Carlo method were developed in statistical physics (the Metropolis method, the Ising model, e.g.), in the physical and chemical kinetics (modeling multi-particle problems, solving the Boltzmann and Smoluchowski's equations, modeling the chemical reactions and phase transitions, etc.), the queuing theory, financial mathematics, turbulence theory, mathematical biology, etc.

The stochastic simulation is thought to be numerical realization of stochastic representation of a certain object in order to estimate its desired integral features with the use of a law of large numbers [12], [13]. We assume that a functional of interest $\varphi \in R$ is represented as expectation of some random variable $\zeta$:

$$\varphi \approx \mathrm{E}\zeta,$$

provided that its variance $\mathrm{Var}\zeta$ is finite. In this case, one can evaluate the value of $\mathrm{E}\zeta$ using a sample mean:

$$\mathrm{E}\zeta \approx \bar{\zeta} = L^{-1} \sum_{i=1}^{L} \zeta_i \tag{1}$$

where $\zeta_i$ are independent realizations of a random variable $\zeta$. The value of $\bar{\zeta}$ is called a **stochastic estimator for $\varphi$**.

One also needs to evaluate the second moment $\mathrm{E}\zeta^2$ of the random variable

$$\mathrm{E}\zeta^2 \approx \bar{\xi} = L^{-1} \sum_{i=1}^{L} \zeta_i^2$$

in order to estimate variance of a random variable $\zeta$ and its standard deviation

$$\mathrm{Var}\zeta \approx \bar{\sigma}^2 = \bar{\xi} - \bar{\zeta}^2,\ (\mathrm{Var}\zeta)^{0.5} \approx \bar{\sigma}.$$

In Monte Carlo methods [12], [13], a complex random variable is represented as a function

$$\zeta = \zeta(\alpha_1, \alpha_2, \dots, \alpha_k), \tag{2}$$

where $\alpha_1, \alpha_2, \dots, \alpha_k$ are independent random variables called **base random numbers** which have uniform distribution on the interval $(0, 1)$. A sequence of the random numbers $\{\alpha_k\}$ is generated with the help of some deterministic numerical algorithm called a **random number generator (RNG)**. Usually, iterative formulas are used [12], [13]:

$$\alpha_{k+1} = f(\alpha_k), k = 0, 1, 2, \dots$$

where $\alpha_0$ is a fixed quantity.

Thus, calculating the realizations $\zeta_i, i = 1, 2, \ldots, L$ we in turn take base random numbers from the RNG. So, to calculate the sample mean we need a finite set of independent random numbers $R = \{\alpha_1, \alpha_2, \ldots, \alpha_S\}$. We call a **stochastic experiment** the process of calculating the sample mean $\bar{\zeta}$ using a particular set of base random numbers $R$. Usually, $R$ is a subsequence of the general sequence $\{\alpha_k\}$ of base random numbers. Using a different set $R' = \{\alpha'_1, \alpha'_2, \ldots, \alpha'_{S'}\}$ (or a different subsequence) of the base random numbers that are independent of the base random numbers from $R$, we finally obtain an independent value of the sample mean. In other words, we carry out the stochastic experiment which is independent of the first one.

A confidence interval of the confidence level $\lambda$ for the expectation $\mathrm{E}\zeta$ is defined by the formula

$$\lambda = \mathrm{P}(|\bar{\zeta} - \mathrm{E}\zeta| \leq \gamma(\lambda)(\mathrm{Var}\zeta)^{0.5}L^{-0.5}) \approx \mathrm{P}(|\bar{\zeta} - \mathrm{E}\zeta| \leq \gamma(\lambda)\bar{\sigma}L^{-0.5}). \quad (3)$$

According to Tables of a standard normal distribution, $\gamma(\lambda) = 3$ for $\lambda = 0.997$. A value of an **absolute (stochastic) error** $\bar{\varepsilon}$ of the stochastic estimator $\bar{\zeta}$ is given by the formula

$$\bar{\varepsilon} = 3(\mathrm{Var}\zeta)^{0.5}L^{-0.5} \approx 3\bar{\sigma}L^{-0.5}$$

and the value of a **relative (stochastic) error** is given by the formula

$$\bar{\rho} = \bar{\varepsilon}/\bar{\zeta} \cdot 100\%.$$

Let us extend the conception of realization of a random object. Assume that at the same time the simulation gives different independent values. It is convenient to represent them as a matrix $[\zeta_{ij}], 1 \leq i \leq n_1, 1 \leq j \leq n_2$. We will also call it a realization (a realization of a random object). After averaging, the following matrices are automatically calculated in the PARMONC:

- $[\bar{\zeta}_{ij}]$ – a matrix of the sample means,
- $[\bar{\varepsilon}_{ij}]$ – a corresponding matrix of the absolute errors,
- $[\bar{\rho}_{ij}]$ – a corresponding matrix of the relative errors,
- $[\bar{\sigma}^2_{ij}]$ – a corresponding matrix of the sample variances.

Also, the following values are automatically calculated: $\bar{\varepsilon}_{max} = \max_{i,j} \bar{\varepsilon}_{ij}$ is the upper bound for the entries of the matrix of the absolute errors; $\bar{\rho}_{max} = \max_{i,j} \bar{\rho}_{ij}$ is the upper bound for the entries of the matrix of the relative errors; $\bar{\sigma}^2_{max} = \max_{i,j} \bar{\sigma}^2_{ij}$ is the upper bound for the entries of the matrix of the sample variances.

In many applications, the above-mentioned matrices and values give an exhaustive information about the stochastic simulation.

## 2.2 Parallelization of Stochastic Simulation

A problem arises when a **computational cost** of the estimator (or computational expenses for obtaining a desired level of the absolute/relative error) is too large. On the average, the computational cost is proportional to the value

$$C(\zeta) = \tau_\zeta \mathrm{Var}\zeta,$$

where $\tau_\zeta$ is a mean computer time to simulate a single realization of $\zeta$. Also, it is clear from formula (3) that the sample volume $L$ needed for obtaining a desired level of accuracy is proportional to the variance Var$\zeta$.

To decrease the computational cost, the simulation of statistically independent realizations may be distributed among $M$ processors (numbered from 0 to $M-1$). At some moment all the processors send subtotal sample means to a dedicated processor (e.g., to 0-th), and the parallel modification of the estimator is given by the formula

$$\bar{\zeta}_M = (\sum_{m=0}^{M-1} l_m)^{-1} \sum_{m=0}^{M-1} l_m \bar{\zeta}^{(m)}, \tag{4}$$

where $l_m$ is a sample volume corresponding to the $m$-th processor, $\bar{\zeta}^{(m)}$ is a corresponding sample mean.

For the massively parallel stochastic simulation, the necessary quantity of base random numbers is very large, and the choice of a parallel RNG must be made with care. For example, a period of a well known RNG with special parameters $r = 40$ and $A = 5^{17}$ is equal to $2^{38} \approx 2.75 \cdot 10^{11}$ (see formulas (6) and (7)) [12], [13]. Such a period is not sufficient for the up-to-date computations: the simulation of a single realization may demand a quantity of base random numbers comparable with the whole period of this generator [5].

Therefore, requirements for a parallel RNG are very rigorous. An important requirement is that sequences of base random numbers $\{\alpha_k\}$ generated on different processors must be independent of each other. Also, base random numbers produced on different processors must have good statistical properties. A necessary information about this subject may be found in [5]. In case of a "good" generator, with increasing the number of processors $M$ and the total sample volume $\sum_{m=0}^{M-1} l_m$, the value of parallel modification (4) goes to E$\zeta$. Naturally, the simulation of realizations must be effectively performed on processors without any problems as those related to memory limitations, etc.

According to this parallelization technique, the stochastic simulation of realizations on different processors is performed in asynchronous mode. It is clear that it is possible to neglect the time expenses for quite rare data exchanges between the 0-th processor and the other ones. In this case, the variance Var$\zeta$ remains the same but the value of $\tau_\zeta$ is decreased. As a result, the value of $\tau_\zeta$ (and, respectively, the value of $C(\zeta)$) is decreased by $M$ times thus giving the optimal parallelization [5].

It is possible to exchange data at the end of simulation when all the processors have simulated the dedicated number of realizations. However, it is not advisable for several reasons. First of all, it is desirable to control the absolute and relative stochastic errors during the simulation. On the other hand, it is useful to create periodic "save-points" of the simulation. For this reason, we modify the parallelization technique in the following way.

Let the $m$-th processor ($m = 0, 1, \ldots, M-1$) periodically sends entries of the matrices $[\bar{\zeta}_{ij}^{(m)}]$ and $[\bar{\xi}_{ij}^{(m)}]$ and the corresponding sample volume $l_m$ (calculated by the moment of sending data) to the 0-th processor. In turn, the 0-th processor periodically receives all the sums $\{\bar{\zeta}_{ij}^{(m)}\}$, $\{\bar{\xi}_{ij}^{(m)}\}$ and the sample volumes $\{l_m\},\ m = 0, 1, \ldots, M-1$, that were sent to it. Then the 0-th processor averages the sample moments:

$$\bar{\zeta}_{ij} = l^{-1} \sum_{m=0}^{M-1} l_m \bar{\zeta}_{ij}^{(m)},\ \bar{\xi}_{ij} = l^{-1} \sum_{m=0}^{M-1} l_m \bar{\xi}_{ij}^{(m)}, \tag{5}$$

where $l = \sum_{m=0}^{M-1} l_m$, calculates the sample variances $\bar{\sigma}_{ij}^2$, the absolute $\bar{\varepsilon}_{ij}$ and the relative $\bar{\rho}_{ij}$ errors of the estimators $\bar{\zeta}_{ij}$. Then the 0-th processor saves the matrices $[\bar{\zeta}_{ij}], [\bar{\varepsilon}_{ij}], [\bar{\rho}_{ij}]$ and $[\bar{\sigma}_{ij}^2]$ in files. Note that the sample volumes $l_m,\ m = 0, 1, \ldots, M-1$ may be different at the moment of passing data. A reason for this fact may consist in different performances of processors or in the diversity of time expenses for the computation of different realizations.

If the frequency of the data exchange with the 0-th processor is not very high, we can neglect the time expenses for the periodical data exchange and averaging. Therefore, the modified parallelization technique enables us to reduce the computational cost of the stochastic simulation nearly by $M$ times. There is also no need to use any load balancing techniques because all the processors work independently and make data exchange in asynchronous mode. This conclusion is proved by an example presented in Subsection 4.

## 2.3 Implementation of Stochastic Simulation

For simplicity let us consider a problem of evaluation of the expectation $\mathrm{E}\zeta$ of a scalar random variable $\zeta$ using the sample mean (1). A typical sequential code (written in C) consists of the following operations:

```
int i, L;
double s, t=0.0;
for(i=0;i<L;i++){
   realization(s);
   t=t+s;
}
t=t/(double)L;
```

Here the argument **L** is the number of independent realizations; **realization** is the name of a sequential routine, which computes a single realization of the random variable $\zeta$ and returns its value to the argument **s**. Finally, the variable **t** gives the value of the sample mean. In the routine **realization** the user calls a function which implements a RNG. The usual use of this function (named **rng()**, e.g.) is as follows:

```
a = rng();
```

Here **a** is the base random number which has the uniform distribution on the interval $(0, 1)$. These numbers are used to simulate necessary complex distributions by formula (2). Given statistically independent outputs from the function **rng()**, all the return values **s** from the subroutine **realization** are statistically independent.

Thereby, a routine, which computes a single realization of a random object, takes the return values from a function that implements a RNG and returns a single realization of a random object. This routine and the specifications for the random object realization are provided by the user. The routine that implements a RNG is considered to be an external routine. In Fig.1 we explain the relationship between main program and data elements in the stochastic simulation.

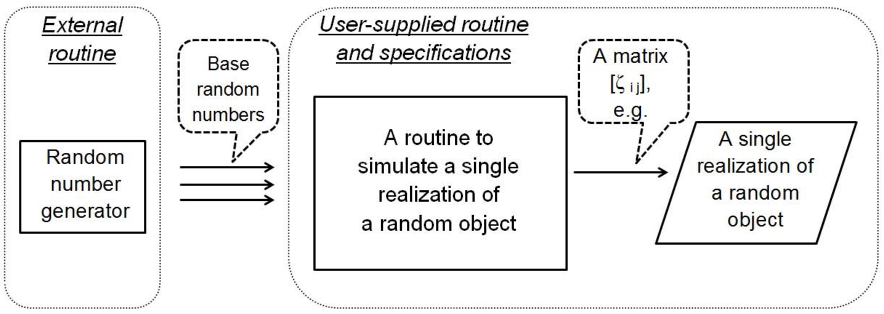
External routine
Random number generator
Base random numbers
User-supplied routine and specifications
A routine to simulate a single realization of a random object
A matrix [ς ij], e.g.
A single realization of a random object

**Fig. 1.** A diagram showing the relationship between the main program and data elements in stochastic simulation

To implement the above-mentioned parallelization technique, the most convenient way is to use a user-defined routine that computes a single realization of a random object as the major piece of the code to be launched on different processors (see Fig. 1). Like in a sequential code, each copy of the routine takes return values from a function that implements the parallel RNG and returns a single realization of a random object. The outputs from all the copies of the user-defined routine (realizations) are taken into account in the course of averaging with the use of formulas (5). This approach is very convenient for specialists in the stochastic simulation because it takes them minimal efforts to adapt their sequential programs for using the PARMONC.

## 2.4 A Parallel RNG

The following base linear congruential generator [12], [13] is used to produce a **general sequence of base random numbers** $\{\alpha_k\}$:

$$u_0 = 1,\ u_{k+1} \equiv u_k A \pmod{2^r},\ \alpha_k = u_k 2^{-r}, \qquad k = 0, 1, 2, \dots . \tag{6}$$

A period of the congruential generator is

$$L_P = 2^{r-2}. \tag{7}$$

We use the following parameters for the generator [14]:

$$r = 128,\ A \equiv 5^{100109} \ (\text{mod } 2^{128}).$$

Therefore, the period of this generator is $2^{126} \approx 10^{38}$. But it is recommended to use the first half of the period only, particularly, the first $2^{125}$ random numbers [12], [13].

In order to obtain independent streams of the base random numbers the general sequence $\{\alpha_k\}$ is divided into subsequences of length $n$ that start with the initial numbers $\tilde{\alpha}_m = \alpha_{nm},\ m = 0, 1, \ldots$. To be exact, the "leaps" of length $n$ are made. **Initial numbers of the subsequences** $\{\tilde{\alpha}_m\}$ are calculated by the formula

$$\tilde{u}_0 = 1,\ \tilde{u}_{m+1} = \tilde{u}_m A(n) \ (\text{mod } 2^r),\ \tilde{\alpha}_m = \tilde{u}_m\ 2^{-r}, \qquad m = 0, 1, 2, \ldots \tag{8}$$

The multiplier $A(n)$ in this auxiliary generator of the "leaps" of length $n$ is calculated as follows:

$$A(n) \equiv A^n (\text{mod } 2^r).$$

This parallel generator enables the convergence of the parallel modification (4) to E$\zeta$. It is implemented as a well tested, fast and reliable routine in the Department of Stochastic Simulation in Physics of the Institute of Computational Mathematics and Mathematical Geophysics in Novosibirsk [15], [16]. It was verified on parallel processors using rigorous statistical testing and solving various problems with known solutions. Therefore, using the parallel generator, we may be sure in the correct stochastic simulation on parallel processors.

The PARMONC parallelization technique is to define a hierarchy of embedded subsequences of the general sequence $\{\alpha_k\}$. The PARMONC assigns subsequences of base random numbers to: a) different stochastic experiments, b) different processors and c) different realizations. The technique is as follows:

– within the general sequence $\{\alpha_k\}$, the "leaps" of length $n_e$ are made using (8) in order to define the initial numbers of subsequences that will be used to perform stochastic experiments (when doing it, **"experiments" subsequences** are produced),
– within each "experiment" subsequence, the "leaps" of length $n_p < n_e$ are made using (8) to define the initial numbers of embedded subsequences that will be used on different processors (when doing it, **"processors" subsequences** are produced),
– within each "processor" subsequence, the "leaps" of length $n_r < n_p$ are made using (8) to define the initial numbers of embedded subsequences that will be used to simulate independent realizations (when doing it **"realizations" subsequences** are produced).

So, the hierarchy of the embedded subsequences is as follows:

general sequence $\supset$ "experiments" subsequences
"experiments" subsequence $\supset$ "processors" subsequences
"processors" subsequence $\supset$ "realizations" subsequences

The initialization of a parallel RNG is as follows: the "experiment" subsequence number is defined by the user with the corresponding argument of the subroutine **parmoncf/parmoncc**; the "processor" subsequence number is automatically defined by the PARMONC with a parallel branch number provided by MPI; the "realizations" subsequence number is automatically defined by the PARMONC before starting the simulation of a realization.

The default lengths of "leaps" are as follows:

- $n_e = 2^{115} \approx 10^{34}$ - for "experiments" subsequences,
- $n_p = 2^{98} \approx 10^{29}$ - for "processors" subsequences,
- $n_r = 2^{43} \approx 10^{13}$ - for "realizations" subsequences.

One can therefore perform approximately $2^{125} \cdot 2^{-115} = 2^{10} \approx 10^3$ stochastic experiments; within a single experiment one can use $2^{115} \cdot 2^{-98} = 2^{17} \approx 10^5$ processors at most and on a processor one can simulate $2^{98} \cdot 2^{-43} = 2^{55} \approx 10^{16}$ independent realizations at most.

In the PARMONC the corresponding generator multipliers $A(n_e), A(n_p)$ and $A(n_r)$ are defined to use by default. Nevertheless, one can redefine the default values of $A(n_e), A(n_p)$ and $A(n_r)$ with the use of the command **genparam** (see Subsection 3.5).

## 3 Overview of the Library PARMONC

A description of the PARMONC can be found on the web site of the Siberian Supercomputer Center [3]; the full description is provided in [4].

### 3.1 Contents of the Library

Contents of the PARMONC is as follows:

- **rnd128** - a function to produce a single base random number,
- **parmoncf** - the main subroutine to perform parallel stochastic simulation (for programs written in FORTRAN),
- **parmoncc** - the main subroutine to perform parallel stochastic simulation (for programs written in C),
- **manaver** - a program to average subtotal sample moments calculated on processors (in a manual mode),
- **genparam** - a program to calculate multipliers of the parallel RNG (in a manual mode).

Here **rnd128, parmoncf and parmoncc** are library routines to use in FORTRAN or C/C++ user-supplied programs, **genparam** and **manaver** are executable files to run from a command line. Object files for the library routines are archived to a static library **libparmonc.a**.

The PARMONC software realization does not use any unique features of a specific FORTRAN compiler or a specific MPI implementation. Therefore, it can be compiled and built with any FORTRAN compiler and MPI library and ported to different high-performance clusters or powerful personal computers with multi-core processors.

### 3.2 Subroutines 'parmoncf' and 'parmoncc'

The subroutine **parmoncf/parmoncc** initializes the parallel RNG, distributes the simulation of independent realizations among processors, makes all operations to pass, to collect and to average data and to save the simulation results in files. The simulation results are stored in several files in a special subdirectory of the user's working directory (see Subsection 3.6).

These subroutines take a name of a user-defined routine which computes a single realization of a random object as argument. The main user-supplied program where a call to **parmoncf/parmoncc** is located is considered as a MPI program despite the fact that it does not contain any MPI instructions (see an example in Section 4). This means that it must be compiled, linked and launched according to specific rules determined by a particular MPI realization on the computer.

The argument **res** is a resumption flag. It defines whether the present simulation resumes the previous one or not:

- **res** = 0 in case of a new simulation. In this case the **parmonc** creates brand new files with results.
- **res** = 1 in case of resuming the previous simulation. In this case the **parmonc** automatically takes into account results of the previous simulation (from the corresponding files) and averages it by formulas (5).

The argument **seqnum** is the "experiments" subsequence number (it is equal to 0,1,2, ...). In case of resuming the previous simulation, this argument must be different from the same argument of the previous use of **parmoncf/parmoncc**.

Also, there are parameters **perpass, peraver** defining the periods of data passing and averaging, respectively, as the number of minutes.

### 3.3 Function 'rnd128'

The double precision function **rnd128** is written using 64-bit integer arithmetic. The function has no arguments. After the initialization of the parallel RNG, **rnd128** starts returning base random numbers from a selected subsequence. Thus, on different processors, parallel streams of base random numbers are generated independently.

### 3.4 Command 'manaver'

The program **manaver** is used to average and to save in files the subtotal sample moments calculated on processors. It is launched after the termination of a job on a cluster. The application of **manaver** is useful in the case when by the moment of terminating the job, the sample moments stored in the files with results correspond to a smaller sample volume than to the one that was actually obtained on all the processors.

### 3.5 Command 'genparam'

If one wants to define different values of the parallel RNG multipliers $A(n_e)$, $A(n_p)$ and $A(n_r)$ in comparison with the default ones, he runs the program **genparam** from a command line in his working directory in the following way:

```
$ genparam ne np nr
```

where **ne, np** and **nr** are exponents of 2. As a result, a file **parmonc_genparam. dat** is created in the user's working directory. Hereupon, the PARMONC routines use the multipliers' values from this file instead of the default ones.

### 3.6 Description of Files with Results of Simulation

When the user launches a job on a cluster, a subdirectory **/parmonc_data** is automatically created by the PARMONC in his/her working directory. In the directory **/parmonc_data/results** one can find the results of computation stored in the files **func.dat**, **func_ci.dat** and **func_log.dat**:

– **func.dat** stores a matrix of the sample means,
– **func_ci.dat** encapsulates a matrix of the sample means together with matrices of absolute and relative errors and variances,
– **func_log.dat** stores information about the stochastic simulation: the total sample volume, the mean computer time per a realization, the upper bounds for absolute and relative errors, etc.

Also, in this directory, one can find a file **parmonc_exp.dat** containing information about each stochastic experiment that was started by the user.

## 4 Performance Test

The following example may be found in the full documentation to the PARMONC [4]. Also, it is available for the users of the the Siberian Supercomputer Center in the directory of the library [3].

We consider a 2-dimensional system of stochastic differential equations (SDEs) over a time interval $[0, 100]$:

$$d\bar{y}(t) = Cdt + Dd\bar{w}(t),$$

where

$$\bar{y}(0) = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \bar{y}(t) = \begin{pmatrix} y_1(t) \\ y_2(t) \end{pmatrix}, C = \begin{pmatrix} 1.0 \\ 1.0 \end{pmatrix}, D = \begin{pmatrix} 10^{-2} & 0 \\ 0 & 10^{-2} \end{pmatrix},$$

and $\bar{w}(t) = \begin{pmatrix} w_1(t) \\ w_2(t) \end{pmatrix}$ is a 2-dimensional Wiener process. Our objective is to evaluate expectations of its components $\mathrm{E}y_1(t)$, $\mathrm{E}y_2(t)$ at fixed points

$t_i = i \cdot 10^{-1},\ i = 1, \dots, 1000$. We simulate trajectories of the SDE system using a generalized Euler method with a mesh size $h = 10^{-6}$:

$$\bar{y}^{(n+1)} = \bar{y}^{(n)} + hC + \sqrt{h}D\bar{\xi}^{(n)}, n = 0, 1, 2, \dots, 10^8, \tag{9}$$

where

$$\bar{y}^{(0)} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \bar{y}^{(n)} = \begin{pmatrix} y_1^{(n)} \\ y_2^{(n)} \end{pmatrix}, \bar{\xi}^{(n)} = \begin{pmatrix} \xi_1^{(n)} \\ \xi_2^{(n)} \end{pmatrix},$$

all *quantity*$\{\xi_i^{(n)}\}$ being independent in total and having a standard normal distribution. The simulation yields a matrix $[\zeta_{ij}]$:

$$\zeta_{ij} = y_j^{(n)}, n = i10^5, 1 \le i \le 1000, 1 \le j \le 2.$$

Thus, each entry of the matrix after averaging gives:

$$\bar{\zeta}_{ij} \approx \mathrm{E}y_j(t_i), t_i = i \cdot 10^{-1},\ i = 1, \dots, 1000, j = 1, 2.$$

Below, as an example, the main user's program in C containing a call to **parmoncc** is provided.

```
int main()
{
   int nrow = 1000, ncol = 2, res = 1, seqnum = 2, perpass = 10,
       peraver = 20;
   long long int maxsv = pow(10,9);
   parmoncc ( difftraj, &nrow, &ncol, &maxsv, &res, &seqnum,
              &perpass, &peraver );
   return 0;
}
```

Here **difftraj** is the name of the user-supplied subroutine implementing the simulation of a realization of an approximate diffusion trajectory according to (9) and returning a realization of matrix $[\zeta_{ij}]$; **nrow** and **ncol** define dimensions of the matrix; **maxsv** is a maximal sample volume to simulate on processors; **res** is a resumption flag; **seqnum** defines the "experiments" subsequence number; **perpass** and **peraver** define the period of sending and receiving data, respectively (in minutes).

In this example **res = 1**. This means the case of resuming the previous simulation: the PARMONC automatically takes into account results of the previous simulation (from the corresponding files) and averages it by formulas (5). Also, **seqnum = 2**. This means that we use the "experiments" subsequence with number 2. Processors send subtotal data to the 0-th processor every 10 minutes. In turn, the 0-th processor receives data every 20 minutes. The argument **maxsv** is chosen to be sufficiently large in order to have an "endless" stochastic simulation that is limited only by the time framework of a job on a cluster (it is defined by the user when starting the job).

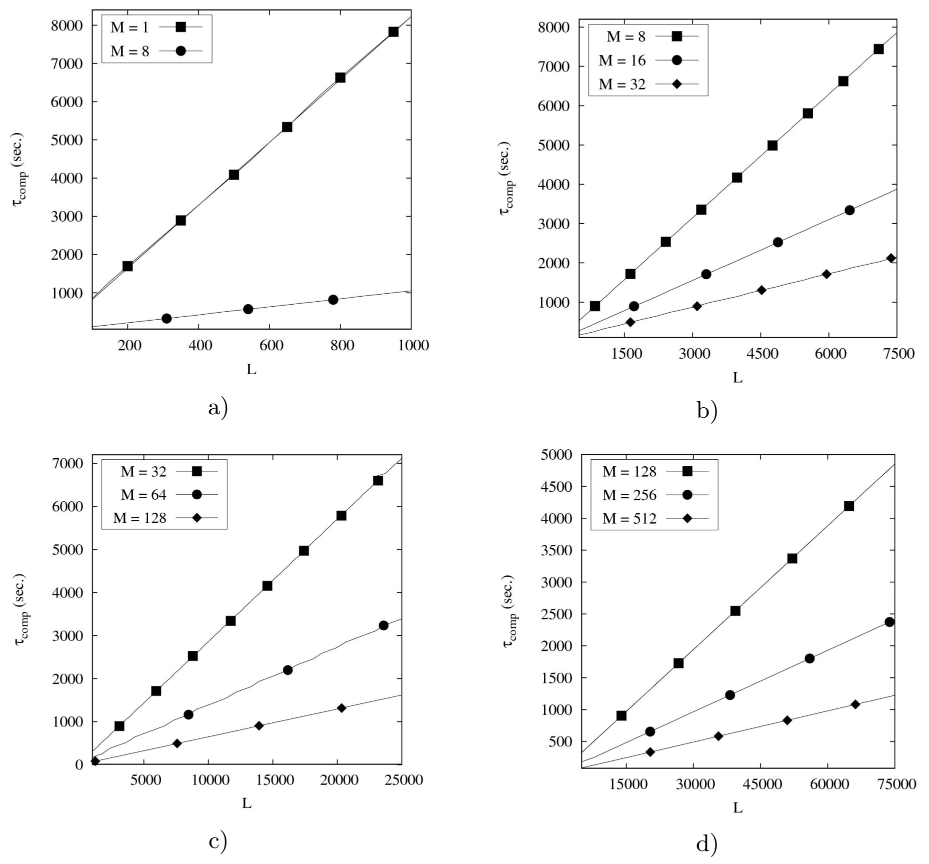

**Fig. 2.** Results of a PARMONC performance test: comparison of the computer time $\tau_{comp} = \tau_{comp}(L)$ for different numbers of processors: a) M = 1 and 8, b) M = 8, 16 and 32, c) M = 32, 64 and 128, d) M = 128, 256 and 512. In each graph X-axis corresponds to the total sample volume $L$, Y-axis corresponds to $\tau_{comp}$ measured in seconds.

In the subroutine **difftraj**, the parallel RNG is called in the following simple way:

```
a = rnd128();
```

This way of calling the RNG seems the most natural for specialists in the stochastic simulation.

The above-mentioned diffusion problem was computed on 1, 8, 16, 32, 64, 128, 256 and 512 processors to compare the speedup of parallelization. All the processors sent data to the 0-th processor after having simulated each realization. In turn, the 0-th processor received data after having simulated each realization. Such conditions are assumed to be strictest in terms of the parallel algorithm performance. A mean computer time $\tau_\zeta$ to simulate a single realization is approximately 7.7 sec., the bulk of data which is periodically sent by every processor to the 0-th processor is approximately 120 Kbytes.

For different numbers of processors $M$ we compare the computer time it takes to simulate $L$ realizations in total $\tau_{comp} = \tau_{comp}(L)$ . A value of $\tau_{comp}$ is evaluated after the 0-th processor has received, averaged and saved data in files.

It is seen from the graphs in Fig. 2 that for all the values of $L$ the speedup of parallelization is in direct proportion to the number of processors despite "strict" conditions related to data exchange.

## 5 Conclusion

In conclusion, we define some directions for the future. First of all, it is desirable to adapt the PARMONC to modern powerful GPU computer clusters and, also, to hybrid computer clusters. Then, it seems promising to use the PARMONC as a basic software level for the future computer-aided simulation based on adequate probabilistic models to imitate real world phenomena from the "first principles".

**Acknowledgements.** The author would like to thank Prof. Victor Malyshkin for our helpful discussions on the subject of this paper; Nikolai Kuchin, Sergey Kireev and Maxim Gorodnichev for their valuable advice; Dr. Galiya Lotova for her patience and help in improving the reliability of the library. Also, the author would like to thank Prof. Boris Glinskiy, Head of the Laboratory "Siberian Supercomputer Center", for his kind attention to the present research.

## References

1. Martin, W.R.: Advances in Monte Carlo Methods for Global Reactor Analysis. In: Invited lecture at the M&C 2007 International Conference, Monterey, CA, USA, April 15-19 (2007)
2. Brown, F.B., Martin, W.R., Mosteller, R.D.: Monte Carlo - Advances and Challenges. In: Workshop at PHYSOR-2008, Interlaken, Switzerland, September 14-19, Report LA-UR-08-05891, Los Alamos National Laboratory (2008), `http://www.physor2008.ch/documents/Workshop_I/PHYSOR08-WorkShopI.pdf`
3. Page of PARMONC on the web site of Siberian Supercomputer Center, `http://www2.sscc.ru/SORAN-INTEL/paper/2011/parmonc.htm`
4. Link to a full documentation to PARMONC, `http://www2.sscc.ru/SORAN-INTEL/paper/2011/parmonc.pdf`
5. Marchenko, M.A., Mikhailov, G.A.: Distributed computing by the Monte Carlo method. Automation and Remote Control 68(5), 888–900 (2007)
6. Brent, R.: Fast and reliable random number generators for scientific computing. In: Dongarra, J., Madsen, K., Waśniewski, J. (eds.) PARA 2004. LNCS, vol. 3732, pp. 1–10. Springer, Heidelberg (2006)
7. The Scalable Parallel Random Number Generators Library (SPRNG), `http://sprng.fsu.edu/`
8. Coddington, P.D., Newell, A.J.: JAPARA – A Java Parallel Random Number Generator Library for High-Performance Computing. In: 18th International Parallel and Distributed Processing Symposium (IPDPS 2004) - Workshop 5, vol. 6, p. 156a (2004)

9. Mendes, B., Pereira, A.: Parallel Monte Carlo Driver (PMCD) - a software package for Monte Carlo simulations in parallel. Comput. Phys. Comm. 151(1), 89–95 (2003)
10. Badal, A., Sempau, J.: A package of Linux scripts for the parallelization of Monte Carlo simulations. Comput. Phys. Comm. 175(6), 440–450 (2006)
11. Slawinska, M., Jadach, S.: MCdevelop - a universal framework for Stochastic Simulations. Comput. Phys. Comm. 182(3), 748–762 (2011)
12. Mikhailov, G.A., Voytishek, A.V.: Numerical stochastic simulation. Publishing Center "Akademia" (2006) (in Russian)
13. Rubinstein, R.Y., Kroese, D.P.: Simulation and the Monte Carlo Method, 2nd edn. John Wiley & Sons, New York (2007)
14. Dyadkin, I.G., Hamilton, K.G.: A study of 128-bit multipliers for congruential pseudorandom number generators. Comput. Phys. Comm. 125(1-3), 239–258 (2000)
15. Marchenko, M.A.: Parallel pseudorandom number generator for large-scale monte carlo simulations. In: Malyshkin, V.E. (ed.) PaCT 2007. LNCS, vol. 4671, pp. 276–282. Springer, Heidelberg (2007)
16. Page of 128-bit parallel congruential random number generator, Department of Stochastic Simulation in Physics of the Institute of Computational Mathematics and Mathematical Geophysics in Novosibirsk, Russia, `http://osmf.sscc.ru/~mam/generator_en.htm`