PARMONC - a software library for massively parallel stochastic simulation

Mikhail Marchenko

May 4, 2011

1 Introduction

You may send the author your questions, advices and remarks using the following emails:

`mam@osmf.sscc.ru, mamarchenko@mail.ru`

The author thanks Sergey Kireev and Maxim Gorodnichev for their advices and helpful discussion.

2 Estimators of interest in stochastic simulation

We assume that a functional of interest $\varphi \in R$ is represented as an expectation of some random variable ζ:

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

One evaluates the value of $\mathrm{E}\zeta$ using a sample mean

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

where sample values ζ_i are independent and identically distributed random variables having the same distribution as ζ. One also needs to evaluate a 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 a variance of the random variable ζ and it's standard deviation

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

A complex random variable my be represented as a function

$$\zeta = \zeta(\alpha_1, \alpha_2, \ldots, \alpha_k),$$

where $\alpha_1, \alpha_2, \ldots, \alpha_k$ are independent random variables (**random numbers**) which have uniform distribution on the interval $(0, 1)$. Therefore, to calculate a sample mean $\bar{\zeta}$ (for some sample volume L) we need a finite set of independent random numbers $R = \{\alpha_1, \alpha_2, \ldots, \alpha_S\}$. We call **a stochastic experiment** a process of calculating the sample mean $\bar{\zeta}$ (for some sample volume L) with a particular set of random numbers R. Using different set $R_1 = \{\alpha_1, \alpha_2, \ldots, \alpha_{S_1}\}$ of random numbers that are independent of the random numbers from R, we get independent value of the sample mean. In other words, we perform the stochastic experiment independent of the first one.

A confidential interval for the expectation $\mathrm{E}\zeta$ is defined by a formula

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

According to tables of 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 a formula

$$\bar{\varepsilon} = 3\bar{\sigma}L^{-0.5}$$

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

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

Assume that a computation of a sample value gives a set of different stochastic estimators at the same time. It is convenient to represent them as a matrix $[\zeta_{ij}], 1 \leq i \leq n_1, 1 \leq j \leq n_2$. In PARMONC the following matrices are automatically calculated:

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

Also, the following values are automatically calculated in PARMONC:

- $\bar{\varepsilon}_{max} = \max_{i,j} \bar{\varepsilon}_{ij}$ – an upper bound for elements of matrix of absolute errors,
- $\bar{\rho}_{max} = \max_{i,j} \bar{\rho}_{ij}$ – an upper bound for elements of matrix of relative errors,
- $\bar{\sigma}^2_{max} = \max_{i,j} \bar{\sigma}^2_{ij}$ – an upper bound for elements of matrix of sample variances.

3 Parallelization of stochastic simulation and random numbers generator

A problem arises when a **computational cost** of the estimator

$$C(\zeta) = t_1 \mathrm{Var}\zeta$$

is too large. Here t_1 is a mean computer time to simulate a sample value. The value of $C(\zeta)$ is proportional to computational expenses to get desired level of the stochastic error.

To decrease computational cost the simulation of statistically independent sample values is distributed among M processors (numbered from 0 to $M-1$). Sample values must be simulated on processors without any problems related to memory limitations, etc. Then it is possible to decrease the computational cost nearly by M times.

On each processor a simulation of the sample values is performed in a loop and this simulation is independent of the simulation on other processors. Periodically, every k-th sample value, the m-th processor ($m = 0, 1, \ldots, M-1$) sends elements of matrices $[\bar{\zeta}_{ij}^{(m)}]$ and $[\bar{\xi}_{ij}^{(m)}]$ calculated by that moment and the corresponding sample volume l_m to the 0-th processor. In turn, periodically, every k_0-th sample value, the 0-th processor receives all sums $\{\bar{\zeta}_{ij}^{(m)}\}$, $\{\bar{\xi}_{ij}^{(m)}\}$ and sample volumes $\{l_m\}$, $m = 0, 1, \ldots, M-1$, that were sent to him. Then it averages the sample moments

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

where $l = \sum_{m=0}^{M-1} l_m$, calculates the sample variances $\bar{\sigma}_{ij}^2$, absolute $\bar{\varepsilon}_{ij}$ and relative $\bar{\rho}_{ij}$ errors of the estimators $\bar{\zeta}_{ij}$. Then it saves matrices $[\bar{\zeta}_{ij}]$, $[\bar{\varepsilon}_{ij}]$, $[\bar{\rho}_{ij}]$ and $[\bar{\sigma}_{ij}^2]$ to a hard disk. It is advisable to make a choice of the parameters k and k_0 in order to get necessary frequency of the data passing and averaging.

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

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

We use the following parameters for the generator:

$$r = 128, \ A \equiv 5^{100109} \pmod{2^{128}}$$

A period of the generator is $2^{126} \approx 10^{38}$.

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

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

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

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

The PARMONC parallelization technique is to assign embedded subsequences of random numbers to a) different stochastic experiments, b) different processors and c) different sample values. The technique is as follows:

- within the general sequence of random numbers the "leaps" of length n_{ex} are made in order to define initial values of subsequences that will be used to perform stochastic experiments (doing it the **"stochastic experiment's subsequences"** are produced),

- within each "stochastic experiment's subsequence" the "leaps" of length $n_{pr} < n_{ex}$ are made to define initial values of embedded subsequences that will be used on different processors (doing it **"processor's subsequences"** are produced),
- within each "processor's subsequence" "leaps" of length $n_{sv} < n_{pr}$ are made to define initial values of embedded subsequences that will be used to simulate sample values (doing it **"sample value's subsequences"** are produced).

An initialization of the parallel random number generator is as follows: a number of the "stochastic experiment's subsequence" is defined by the user with the corresponding argument of the subroutine **parmonc**; a number of the "processor's subsequence" is defined automatically by PARMONC with a number of a parallel branch provided by MPI; a number of the "sample values'" subsequence is defined automatically by PARMONC before starting the simulation of the sample value.

Default lengths of "leaps" are as follows:

- $n_{ex} = 2^{106} \approx 10^{32}$ - for "experiment's subsequences",
- $n_{pr} = 2^{86} \approx 10^{26}$ - for "processor's subsequences',
- $n_{sv} = 2^{53} \approx 10^{16}$ - for "sample value's subsequences".

One can therefore perform approximately $10^{38} \cdot 10^{-32} = 10^6$ stochastic experiments; for a single experiment one can use $10^{32} \cdot 10^{-26} = 10^6$ processors maximum and on a processor one can simulate $10^{26} \cdot 10^{-16} = 10^{10}$ sample values maximum.

In PARMONC the corresponding generator multipliers $A(n_{ex}), A(n_{pr})$ and $A(n_{sv})$ are defined to use by default. Nevertheless, one can redefine the default values of $A(n_{ex}), A(n_{pr})$ and $A(n_{sv})$ with the use of the command **genparam**.

4 Overview of the library PARMONC

4.1 Contents of the library

Contents of the PARMONC is as follows:

- **rnd128** - a function to produce a single random number,
- **parmoncf** - a main subroutine to perform parallel stochastic simulation (for Fortran),
- **parmoncc** - a main subroutine to perform parallel stochastic simulation (for C),
- **manaver** - a program to average subtotal sample moments calculated on processors (in a manual mode),
- **genparam** - a program to calculate generator's parameters (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**.

4.2 Some features of the library

A user inserts calls to **rnd128** and **parmoncf/parmoncc** to his programs. The following INCLUDE directive must be inserted to a main C program :

```
#include "parmonc.h"
```

A 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. It means that it must be compiled, linked and launched according to specific rules determined by a particular MPI realization on the computer.

Results of simulation performed by **parmoncf/parmoncc** are stored in a user working directory in several files. All these files are updated periodically each time the 0-th processor receives data from processors, averages it and saves to a hard disk.

A directory **/parmonc_data/service** contains necessary auxiliary files. Each time the processors send data to the 0-th processors they save files with the subtotal sample means in the specific format to a directory **/parmonc_data/service/subtotal/processors**. Each time the 0-th processor saves data to a hard disk it also saves the auxiliary file with results of averaging in the specific format to a directory **/parmonc_data/service/subtotal/experiments**. These files are used when the simulation is resumed after its' previous termination. These files are also used by the program **manaver** for the same goal.

It is necessary to add the following string to the file **.bashrc** that is located in user's home directory:

```
source /ifs/apps/parmonc/bin/parmoncvars.sh
```

As a result, three system environment variables \$PRMCBIN, \$PRMCLIB and \$PRMCINC are defined. These variables are used to compile and link programs with PARMONC and to start PARMONC executable files. Contents of the file **parmonvars.sh** is as follows:

```
PRMCBIN=/ifs/apps/parmonc/bin; export PRMCBIN
PRMCLIB=/ifs/apps/parmonc/lib; export PRMCLIB
PRMCINC=/ifs/apps/parmonc/inc; export PRMCINC
export PATH=$PATH:/ifs/apps/parmonc/bin
```

4.3 Function 'rnd128'

The function has no arguments. After the initialization of the parallel random number generator **rnd128** starts returning random numbers from the selected subsequence. Thus, on different processors parallel streams of random numbers are generated independently.

Listing 3.1.1. *Definition of* **rnd128**

```
real*8  function rnd128()
```

4.4 Subroutines 'parmoncf' and 'parmoncc'

Below a declaration of the subroutine **parmoncf** is given. A declaration of the subroutine **parmoncc** is the same.

Listing 3.2.1. *Declaration of* **parmoncf**

```
subroutine parmoncf(fsample, numrow, numcol, numruns,&
                     resume, expseqnum, peraverage, perpass)
integer*4 numrow, numcol, numruns, resume, expseqnum, &
              peraverage, perpass
external fsample
```

A description of its arguments is as follows:

- **fsample** – name of a user-supplied subroutine that calculates a sample value (INPUT).

  A declaration of **fsample** in Fortran is as follows:

  ```
  subroutine fsample ( numrow, numcol, sample )
  integer*4 numrow, numcol
  real*8 sample(numrow, numcol)
  ```

  where

  - **numrow** – number of rows in a matrix (integer, INPUT)
  - **numcol** – number of columns in a matrix (integer, INPUT)
  - **svalue** – a 2-dimensional matrix (double precision, OUTPUT)

  A declaration of **fsample** in C is as follows:

  ```
  void fsample( int *numrow, int *numcol, double *svalue )
  ```

  - **numrow** – number of rows in a matrix (pointer to integer value, INPUT)
  - **numcol** – number of columns in a matrix (pointer to integer value, INPUT)
  - **svalue** – a 2-dimensional matrix (pointer to double precision value, OUTPUT)

- **numrow** – number of rows in a sample value matrix (integer, INPUT)
- **numcol** – number of columns in a sample value (integer, INPUT)
- **numruns** – maximal number of sample values to simulate (integer, INPUT)
- **resume** – resumption flag (integer, INPUT).

  It defines whether the present simulation resumes the previous one or not.

  - **resume** = 0 in the case of a new simulation. In that case the **parmonc** creates brand new files with results,

  - **resume** = 1 in a case of resuming the previous simulation. In that case the **parmonc** automatically takes into account results of the previous simulation (from corresponding files) and averages it as previously described. A necessary verification is performed automatically.

- **expseqnum** – number of subsequence for a stochastic experiment (integer, INPUT).

  In a case of resuming the previous simulation this argument must be different of the same argument of the previous use of **parmonc**.

- **peraverage** – a period value to make averaging task (integer, INPUT).

  The 0-th processor collects and averages subtotal data from processors every **peraverage**-th sample value.

- **perpass** – a period value to pass the data (integer, INPUT).

  Each processor passes subtotal data every **perpass**-th sample value.

4.5 Command 'manaver'

The program **manaver** is used to average and save to a hard disk the subtotal sample moments calculated on processors. It is launched after the termination of the job on the cluster. It is useful in a 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 the one that was actually obtained on all processors. Launching **manaver** from the command line in his working directory

```
$ manaver
```

a user forces PARMONC to read and average files with results obtained by processors that are stored in a subdirectory **/parmonc_data/service/subtotal/processors** of his working directory. The program uses the information about last stochastic experiment that is stored in the file **/parmonc_data/service/newexp.log**. The program automatically takes into account the value of the resuming flag.

4.6 Command 'genparam'

If one wants to define different values of generator multipliers $A(n_{ex}), A(n_{pr})$ and $A(n_{sv})$ to use different values of "leaps" lengths n_{ex}, n_{pr} and n_{sv}, he runs the program **genparam** from a command line in his working directory:

```
$ genparam e p s
```

where arguments **e, p, s** are exponents to 2 that define lengths of "jumps" for parallel generator ($126 > e > p > s$):

$$2^e = n_{ex}, 2^p = n_{pr}, 2^s = n_{sv}.$$

As a result, a file **parmonc_genparam.dat** is created in the user's working directory. A presence of this file orders PARMONC routines to use the multipliers' values from the file instead of the default values.

For example, in PARMONC the default values of "leaps" lengths are $n_{ex} = 2^{106}, n_{pr} = 2^{86}$ and $n_{sv} = 2^{53}$. The corresponding values of multipliers may be obtained by running the following command:

```
$ genparam 106 86 53
```

4.7 Description of files with results of simulation

When user runs his program, a subdirectory **/parmonc_data** is created automatically in his working directory. Below a typical content of a working directory is provided (the listing was shortened).

Listing 3.5.1. *Contents of the working directory displayed as a tree:*

```
/user_work_dir
.    /parmonc_data
.    .   /results
.    .   .    func.dat
.    .   .    func_ci.dat
.    .   .    func_log.dat
.    .   .    parmonc_exp.dat
.    .   /service
.    .   .    /subtotal
.    .   .    .    /experiments
.    .   .    .    .    func_exp_000000
.    .   .    .    .    func_exp_000001
.    .   .    .    .    . . .
.    .   .    .    /processors
.    .   .    .    .    func_pr_000000
.    .   .    .    .    func_pr_000001
.    .   .    .    .    . . .
.    .   .    newexp.log
.    . . .
.    diffusion
.    diffusion_sh
.    diffusion.f90
.    diffusion.c
.    parmonc_genparam.dat
```

In a directory **/parmonc_data/results** one can find the results of computation stored in files **func.dat**, **func_ci.dat** and **func_log.dat**:

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

Also, in this directory one can found a file **parmonc_exp.dat** containing information about parameters of stochastic experiments carried out.

A directory **/parmonc_data/service** contains necessary auxiliary files. In directories **/parmonc_data/service/subtotal/experiments** and

/parmonc_data/service/subtotal/processors subtotal sample moments corresponding to different stochastic experiments and processors are stored in separate files. Each time the processors send data to the 0-th processors they save files with the subtotal sample means in the specific format. Each time the 0-th processor saves data to a hard disk it also saves the auxiliary file with results of averaging in the specific format. These files are used by the program **manaver**.

5 Example of PARMONC application

Below we describe typical steps how to: a) write a typical application for stochastic simulation, b) compile it with the use of the PARMONC, c) launch it on the cluster, d) postprocess the results of the simulation, e) analyze results of the simulation, f) resume the simulation after terminating.

The example is provided on the cluster NKS-30T in the directory **/ifs/apps/parmonc/examples/diffusion** (there are different directories for FORTRAN and C examples).

5.1 Description of a problem

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

$$d\bar{y}(t) = Cdt + Dd\bar{w}(t), \bar{y}(t) = 0$$

where

$$\bar{y}(t) = \begin{pmatrix} y_1(t) \\ y_2(t) \end{pmatrix}, C = \begin{pmatrix} 1.0 \\ 1.0 \end{pmatrix}, D = \begin{pmatrix} 0.01 & 0 \\ 0 & 0.01 \end{pmatrix},$$

and $\bar{w}(t) = \begin{pmatrix} w_1(t) \\ w_2(t) \end{pmatrix}$ is a 2-dimensional Wiener process.

A goal is to evaluate expectations of its components $\mathrm{E}y_1(t),\ \mathrm{E}y_2(t)$ at fixed points $t_i = i \cdot 10^{-1},\ i = 1, \ldots, 1000$. We simulate trajectories of the SDE system using a generalized Euler method with a stepsize $h = 10^{-6}$:

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

where

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

all $\xi_i^{(n)}$ being independent in total and having standard normal distribution. A simulation yields a matrix $[\zeta_{ij}], 1 \leq i \leq 1000, 1 \leq j \leq 2$:

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

Thus, each element of the matrix gives after averaging

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

5.2 Writing a user-defined program in FORTRAN

Below a user-defined program **diffusion.f90** is provided.

Listing 4.2.1 *Main program where* **parmoncf** *is called*

```
program diffusion

implicit none
integer*4 numrow, numcol, maxsampvol, resuming, expseqnum, &
 perpass, peraverage
external difftraj

numrow = 1000
numcol = 2
maxsampvol = 10**9
resuming = 0
expseqnum = 0
perpass = 10
peraverage = 20

call parmoncf (difftraj, numrow, numcol, maxsampvol, resuming, expseqnum, &
 perpass, peraverage)

end
```

Note that in this code we set **resuming = 0**. We make it because we start a new simulation. Also, we set **expseqnum = 0**. Thus, we choose the first (with the number 0) experiments' subsequence.

Listing 4.2.2 *Listing of a subroutine* **difftraj** *returning a sample value and other routines where calls to* **rnd128** *are located*

```
subroutine difftraj ( num_row, num_col, sample )

implicit none
integer*4 num_row, num_col
real*8 sample(num_row, num_col)
integer i, k, isave
real*8 deltat, y(2), tau, tend

tend = 1.0d2
deltat = 1.0d-6

isave = 10**5

y(1) = 0.0d0
y(2) = 0.0d0

tau = 0.0d0
i = 0
```

```
k = 0

do while (1)
call euler_method (deltat, y)
tau = tau + deltat
i = i + 1
if (tau.gt.tend) then
exit
endif
if ( mod(i, isave) == 0 ) then
k = k + 1
sample(k, 1) = y(1)
sample(k, 2) = y(2)
endif
enddo

return
end
!*********
subroutine euler_method (deltat, y)

implicit none
real*8 deltat, y(2)
real*8 d(2, 2)
real*8 c(2), ksi(2)
integer i, j

call normdistr2( ksi )
call convection( c )
call diffusion( d )

do i = 1, 2
y(i) = y(i) + deltat * c(i)
do j = 1, 2
y(i) = y(i) + dsqrt(deltat) * d(i, j) * ksi(j)
enddo
enddo

end
!*********
subroutine normdistr2 ( ksi )

implicit none
real*8 ksi(2)
real*8 r1, r2, r3, e1, pi
real*8, external :: rnd128

pi = 3.1415926535897932d0
r1 = rnd128()
r2 = rnd128()
```

```
e1 = dsqrt( -2.0d0 * dlog(r1) )
r3 = 2.0d0 * pi * r2
ksi(1) = e1 * dsin( r3 )
ksi(2) = e1 * dcos( r3 )

end
!*********
subroutine convection ( c )
implicit none
real*8 c(2)
c(1) = 1.0d0
c(2) = 1.0d0
return
end
!*********
subroutine diffusion ( d )
implicit none
real*8 d(2, 2)
d(1, 1) = 1.0d-2
d(1, 2) = 0.0d0
d(2, 1) = 0.0d0
d(2, 2) = 1.0d-2
end
```

5.3 Writing a user-defined program in C

Below a user-defined program **diffusion.c** is provided.

To write a code in C it is necessary to take into account some specific features of this programming language:

- Calls to library routines are made with the underscore character at the end: **parmoncc_** and **rnd128_**.
- Be careful when you deal with the output argument for a sample value (a matrix) in the routine that simulates a sample value. At first, a type casting for the input pointer must be made. Also, an assignment of sample value components must be made in accordance to the specific rules. Look at the example with the **difftraj** routine.

Listing 4.3.1 *Main program where* **parmoncc_** *is called*

```
int main()
{
int numrow, numcol, maxsampvol, resuming, expseqnum,
                perpass, peraverage;

numrow = 1000;
numcol = 2;
maxsampvol = pow(10,9);
resuming = 0;
expseqnum = 0;
perpass = 10;
peraverage = 20;

parmoncc_ (difftraj, &numrow, &numcol, &maxsampvol, &resuming, &expseqnum,
              &perpass, &peraverage);
}
```

Note that in this code we set **resuming = 0**. We make it because we start a new simulation. Also, we set **expseqnum = 0**. Thus, we choose the first (with the number 0) experiments' subsequence.

Listing 4.3.2 *Listing of a routine* **difftraj** *returning a sample value and other routines where calls to* **rnd128_** *are located*

```
void difftraj (int *numrow, int *numcol, double *sample1d)
{
// BE CAREFUL: assign new variables with values of pointers which define matrix dimensions

int nr=*numrow, nc=*numcol;

// BE CAREFUL: define and initialize a new variable and make type casting for input matrix
// (*sample) is a matrix to put sample values in

double (*sample)[nr][nc]=(double (*)[nr][nc])sample1d;

int i, k, isave;
double h, tau, tend, x[2];

tend = 1.0e2;
h = 1.0e-6;
isave = pow(10,5);

x[0] = 0.0;
x[1] = 0.0;

tau = 0.0;
i = 0;
k = 0;

do
{
Euler_Method (h, x);
tau += h;
i++;
if (tau >= tend) break;
if ( i%isave == 0 )
{

// BE CAREFUL: assigning the output matrix

(*sample)[k][0] = x[0];
(*sample)[k][1] = x[1];

k++;
};
} while(1);
}
//********
void normdistr2d ( double ksi[2] )
{
double r1, r2, r3, e1, pi = 3.1415926535897932;

r1 = rnd128_();
r2 = rnd128_();

e1 = sqrt( -2.0 * log(r1) );
r3 = 2.0 * pi * r2;
ksi[0] = e1 * sin( r3 );
ksi[1] = e1 * cos( r3 );
}
//********
void convection ( double f[2] )
{
f[0] = 1.0;
f[1] = 1.0;
}
//********
void diffusion ( double (*s)[2] )
{
s[0][0] = 1.0e-2;
s[0][1] = 0.0;
s[1][0] = 0.0;
s[1][1] = 1.0e-2;
}
//********
void Euler_Method (double deltat, double x[2])
{
double f[2], ksi[2], s[2][2];
int i, j;

normdistr2d( ksi );
convection( f );
diffusion( s );

for(i=0;i<2;i++)
{
x[i] += deltat * f[i];
for(j=0;j<2;j++){
x[i] += sqrt(deltat) * s[i][j] * ksi[j];
}
}
```

```
}
```

5.4 Compiling the user-defined program

To compile user-defined program and build it using PARMONC, one should launch the following commands:

for programs written in FORTRAN:

```
$ mpiifort -o diffusion -L$PRMCLIB -I$PRMCINC diffusion.f90 -lparmonc
```

for programs written in C:

```
$ mpiicc -o diffusion -L$PRMCLIB -I$PRMCINC diffusion.c -lparmonc
```

Here **diffusion** is the name of the executable file, **diffusion.f90** and **diffusion.c** are the user-defined programs.

5.5 Starting a job on the cluster

To launch a simulation one should start a job on the cluster. To do this, the starting script must be launched. An example of such script (named **parmonc_sh**) is listed below.

Listing 4.5.1. *Example of the starting script*

```
#!/bin/bash
#PBS -V
#PBS -q g6_q
#PBS -r n
#PBS -l nodes=2:ppn=8,cput=04:10:00,walltime=04:15:00
#PBS -N diffusion
#PBS -j oe
date
cd $PBS_O_WORKDIR
pwd
MPD_CON_EXT='date'
##cat $PBS_NODEFILE
mpirun -r ssh  -genv I_MPI_FABRICS shm:ofa -n 16 ./diffusion
date
```

Here we use 2 nodes of the cluster NKS-30T (1 node = 2 processors = 2*4 = 8 cores). Thus, for PARMONC the number of processors equals to 16.

5.6 Postprocessing: applying command 'manaver'

To take into account the total sample volume that was obtained on all processors it is advisable to launch command **manaver** from the command line in the user's working directory

```
$ manaver
```

Doing it the user forces PARMONC to read and average files with results obtained by processors that are stored in a directory **/parmonc_data/service/subtotal/processors** in his working directory.

5.7 Analyzing results of the stochastic simulation

A few lines from the file **func.dat** are given. First line is a file header with names of columns, other lines contain calculated quantities that corresponds to elements of the matrix $[\bar{\zeta}_{ij}]$.

Listing 4.7.1. *Shortened listing of the file* **func.dat**

```
    est001,         est002
    0.20582E-04,    0.14813E-04
    0.26291E-04,    0.30634E-04
    0.30480E-04,    0.18184E-04
    0.32538E-04,    0.22777E-04
...
```

That is, $\mathrm{E}y_1(t_1) \approx 0.20582 \cdot 10^{-4}$, $\mathrm{E}y_2(t_1) \approx 0.14813 \cdot 10^{-4}$; $\mathrm{E}y_1(t_2) \approx 0.26291 \cdot 10^{-4}$, $\mathrm{E}y_2(t_2) \approx 0.30634 \cdot 10^{-4}$, etc.

A few lines from the file **func_ci.dat** are given. First line is a file header with names of columns, other lines contain calculated quantities (only the first bunch of columns is given).

Each bunch of columns with names

```
est<k>, abserr<k>, relerr<k>, var<k>, where <k> is either 1 or 2
```

corresponds to a bunch of k-th columns of the matrices $[\bar{\zeta}_{ij}], [\bar{\varepsilon}_{ij}], [\bar{\rho}_{ij}], [\bar{\sigma}^2_{ij}]$, respectively.

Listing 4.7.2. *Shortened listing of the file* **func_ci.dat**. *Right bunch of columns is cut off*

```
    est001,         abserr001,          relerr001,          var001, ...
    0.20582E-04,    0.58925E-04,    0.28628E+03,    0.99575E-05, ...
    0.26291E-04,    0.83174E-04,    0.31635E+03,    0.19839E-04, ...
    0.30480E-04,    0.10182E-03,    0.33407E+03,    0.29735E-04, ...
 ...
```

That is, $\bar{\zeta}_{11} \approx 0.20582 \cdot 10^{-4}, \bar{\varepsilon}_{11} \approx 0.58925 \cdot 10^{-4}, \bar{\rho}_{11} \approx 286.3\%, \bar{\sigma}^2_{11} \approx 0.995751 \cdot 10^{-5}$, etc. Thus, $\mathrm{E}y_1(t_1) \approx 0.20582 \cdot 10^{-4} \pm 0.58925 \cdot 10^{-4}$, etc.

Below a listing of the file **func_log.dat** is given.

Listing 4.7.3. *Listing of the file* **func_log.dat**

```
    0.11100E+05     total sample volume
    0.10877E+02     mean computational time per sample in sec.
    0.28586E-02     absolute error upper bound
    0.16671E+05     relative error upper bound in %
    0.10078E-01     variance upper bound
```

That is, in total 11100 sample values were simulated, a mean computational time consumed to simulate a sample on a processor is 10.88 sec., the absolute error upper bound is $\bar{\varepsilon}_{max} = 0.28586 \cdot 10^{-2}$, the relative error upper bound is $\bar{\rho}_{max} = 16671\%$, the variance upper bound is $\bar{\sigma}^2_{max} = 0.10078 \cdot 10^{-1}$.

When the job is started, the contents of the file **parmonc_exp.dat** is as follows:

Listing 4.7.4. *Listing of the file* **parmonc_exp.dat**

```
Started Wed Apr 20 12:12:52 2011:
     number of processors =         16
     resuming = no
     number of experiment‘s subsequence =      0
     passing data period =         10
     averaging period =         20
     max.sample volume =      1000000000
```

This data represents main parameters characterizing stochastic experiment.

5.8 Resuming simulation after terminating

To resume simulation previously performed and terminated it is necessary to set the arguments of **parmonc** as described below.

Listing 4.8.1 *Setting arguments of the* **parmonc** *in order to resume simulation*

```
program diffusion

implicit none
integer*4 numrow, numcol, maxsampvol, resuming, expseqnum, &
 perpass, peraverage
external fsample

numrow = 1000
numcol = 2
maxsampvol = 10**9
resuming = 1
expseqnum = 1
perpass = 10
peraverage = 20

call parmonc (fsample, numrow, numcol, maxsampvol, resuming, expseqnum, &
 perpass, peraverage)

end
```

We changed the argument **resuming** from 0 to 1 (doing it we allow resuming of the simulation). Also, we changed the argument **expseqnum**. The argument **expseqnum** representing the number of experiments' sequence, when we want to perform different stochastic experiment we increase the value of **expseqnum** by 1 in comparison with the previous experiment.

Arguments **maxsampvol, perpass, peraverage** may be changed at the discretion of the user.

Note, that it is important **not to change arguments 'numrow' and 'numcol'**, otherwise the result of computing and averaging may be unpredictable.

When the job is started, the contents of the file **parmonc_exp.dat** becomes as follows:

Listing 4.8.2 *Listing of the file* **parmonc_exp.dat**

```
Started Wed Apr 20 12:12:52 2011:
     number of processors =         16
     resuming = no
     number of experiment‘s subsequence =      0
     passing data period =         10
     averaging period =         20
     max.sample volume =      1000000000
Started Wed Apr 20 14:34:43 2011:
     number of processors =         32
     resuming = yes
     number of experiment‘s subsequence =      1
     passing data period =         10
     averaging period =         20
     max.sample volume =      1000000000
```

If we perform another two iterations the contents of the file becomes the following:

Listing 4.7.3 *Listing of the file* **parmonc_exp.dat**

```
Started Wed Apr 20 12:12:52 2011:
     number of processors =         16
     resuming = no
     number of experiment‘s subsequence =      0
     passing data period =         10
     averaging period =         20
     max.sample volume =      1000000000
Started Wed Apr 20 14:34:43 2011:
     number of processors =         32
     resuming = yes
     number of experiment‘s subsequence =      1
     passing data period =         10
     averaging period =         20
     max.sample volume =      1000000000
Started Wed Apr 20 15:05:56 2011:
     number of processors =          8
     resuming = yes
     number of experiment‘s subsequence =      2
     passing data period =         10
     averaging period =         20
     max.sample volume =      1000000000
Started Wed Apr 20 17:49:30 2011:
     number of processors =          8
     resuming = yes
     number of experiment‘s subsequence =      3
     passing data period =         10
     averaging period =         20
     max.sample volume =      1000000000
```

References

[1] Marchenko, M.A., Mikhailov, G.A.: Distributed computing by the Monte Carlo method. Automation and Remote Control, vol. 68, issue 5, pp. 888–900 (2007)

[2] Marchenko, M.A.: Parallel Pseudorandom Number Generator for Large-scale Monte Carlo Simulations. In: Malyshkin, V. (Ed.) PaCT 2009. LNCS, vol. 4671, pp.276–282 (2007)

[3] Page of PARMONC in a directory of Foundation of Algorithms and Programs of SB RAS, `http://fap.sbras.ru/node/1564`

[4] Page of 128-bit parallel congruential random number gennerator, 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`