Computing and Informatics, Vol. 30, 2011, 681–700

THE USE OF REFINED DESCRIPTIVE SAMPLING AND APPLICATIONS IN PARALLEL MONTE CARLO SIMULATION

Abdelouhab Aloui, Megdouda Ourbih-Tari

Laboratory of Applied Mathematics
University of Bejaia
Algeria

Communicated by Ivan Plander

Abstract. Refined descriptive sampling is designed to improve upon the descriptive sampling method for experimentation in simulation. The former reduces significantly the risk of sampling bias generated by descriptive sampling and eliminates its problem related to the sample size. In this paper, we propose an optimal parallel Monte Carlo simulation algorithm using refined descriptive sampling and evaluate in parallel architecture, performance measures of a stable M/M/1 queueing system, a Pert network and the Newsboy problem.

Keywords: Monte Carlo, sampling, parallel simulation, queuing system, Pert network, Inventory

Mathematics Subject Classification 2000: 62D05, 90B05, 90B22, 90B30

1 INTRODUCTION

Monte Carlo method also called random sampling (RS) can solve a large variety of problems, but both kinds of variations (set effect and sequence effect) are present in a randomly generated sample that provides an imprecise evaluation of each simulation estimates (which is a function of the input values). To remedy the lack of accuracy that is related to the method itself, a new paradigm has emerged. The latter says that is not always necessary to generate sample values randomly to describe a stochastic behavior. Then, new deterministic methods, like Quasi Monte

Carlo [24], Latin hypercube sampling [18], Descriptive Sampling (DS) [29] as well as Refined Descriptive Sampling (RDS) [35] were derived from this paradigm.

RDS is based on a deterministic selection of the input sample values. Subsets of regular numbers are first generated, then randomly shuffled and finally observations of the input random variables are generated as required by the simulation since the dimension of subsets are prime numbers randomly generated. This method was proposed by Tari [34] to make DS safe, efficient and convenient; safe by reducing substantially the risk of sampling bias, efficient by producing estimates with lower variances and convenient by removing the need to determine in advance the sample size.

On the other hand, simulation modelling involves both repetition and iteration. A large simulation model may take a number of hours to run and, of course, many runs may be required for thorough experimentation. Because simulation is time-consuming approach, it is recommended to be used as a means of last resort, rather than the preferred option [25]. Indeed, simulation studies systems whose analytical solution is difficult even impossible. Surveys of modelling practice demonstrate that simulation is one of the most commonly used modelling techniques of systems performance measures evaluation. Most simulation software is not cheap and most simulation projects take few weeks to complete. Few software applications require the computing power necessary for simulation. Even Taylor et al. [39] explore the use of net conferencing during simulation studies. It is then obvious that a simulation study is expensive in time, memory and more generally in necessary resources [1]. Indeed, the traditional simulation tools could be unusable when complex systems are studied. We are then confronted with the problem of the best compromise between the precision of simulation estimates and the cost of the experiment. The latter of course must be balanced against the benefits that can be gained from the use of simulation, in the same way, the precision of the estimates that are often an order of magnitude greater than the cost. Unfortunately, the limited means of available sequential computation make achieving this goal difficult. Therefore, a parallel simulation procedure is needed for solving the above problem. Parallelism is regarded as a means of reducing the cost in time and memory of a program solving a complex problem.

We can find several works on the parallelization of Monte Carlo methods but all kinds of variations are still present in a randomly generated sample and simulation estimates remain affected by such sampling errors through Monte Carlo parallel algorithms already proposed for example in [2, 22, 40].

In this paper, the best sampling procedure – RDS is selected to improve – the accuracy of simulation estimates and its safe implementation is suggested in parallel architecture to reduce the cost of running simulation experiments. However, RDS procedure is regarded as an algorithm of dependent instructions capable to be executed just in sequential way. The only instruction able to be parallelized is related to the loop of the array filling up with regular numbers but it is not well-suited to parallelization because of the high communication cost that may be generated. However, MC simulation using RDS involves repetition and this feature is specially well-suited to parallelization. Then, to avoid such communication cost, parallelism of the number of replicated runs is proposed using RDS to generate input distributions. We have then carried out a parallel Monte Carlo simulation program running on machines with distributed memory. Accordingly, the Single Program Multiple Data (SPMD) model has been selected for a better programming related to the replicated runs and the Message Passing Interface (MPI) library [11] has been suitably chosen to be used with such model. The selected library supports both C/C++ and FORTRAN and allows data exchange between the processors. The selected library fits well our requirements; nevertheless we can find other libraries like Parallel Virtual Machine (PVM) [13], Portable Programs for Parallel Processors (P4) [4] and Open Multi Processing (OpenMP) [6].

The use of SPMD model provides an optimal efficiency of the proposed parallel algorithm. This paper also evaluates in parallel architecture performance measures of some problems such as a stable M/M/1 queueing system, Pert network and Newsboy problem. In the designed parallel software[1], we used RDS algorithm given in [36] to generate input refined descriptive sample values and the built in generator of the computing environment to generate both prime numbers and integers to permute the regular numbers related to RDS method[2].

2 ON CHOOSING THE BEST SAMPLING PROCEDURE

Monte Carlo [9] is a numerical method of solving problems using random numbers and it is one of the most popular mathematical tools with a very large application field; like differential equation integration, matrix inversion, particles transport, fluids mechanic and financial mathematics [5, 8, 26, 28]. The following sampling methods are all regarded as variance reduction techniques.

2.1 Quasi Monte Carlo Method

In quasi Monte Carlo methods, we use deterministic sequences of weak convergence to reduce the variance in MC method. In other words, we replace the random points by a set of points that cover the unit hypercube more uniformly than typical random points. The two main classes of methods for constructing such point sets are digital nets and integration lattices [17, 20, 33].

Okten, Owen and Tuffin [21, 23, 40], using random sampling with low discrepancy sequences, randomized nets and sequences and a random permutation of QMC methods, respectively carried out various comparisons between the simulation of MC and QMC producing hybrid methods. The existing literature in this field abounds of theoretical results on QMC, for example [3].

[1] The simulation programme of each application is written by using the C++ compiler and run on Pentium 4 under the operating system Linux.

[2] The built in pseudo random number generator used is the rand() function of the C++ compiler under Linux.

2.2 Latin Hypercube Sampling

In 1979, McKay et al. [19] proposed Latin Hypercube Sampling (LHS) as a variance reduction technique in which the selection of sample values is highly controlled, although still letting them to vary. The basis of LHS is a full stratification of the sampled distribution with a random selection inside each stratum. It is a kind of stratified random sampling where sample values are randomly shuffled among different variables. Some theoretical results on LHS can be found, for example in Hoshino and Takemura [14].

2.3 Descriptive Sampling

DS is based on a deterministic selection of the input sample values and their random permutation. This method was introduced by Saliby in 1990 [29] as a better alternative to Monte Carlo simulation avoiding the variability of the sample values that is an undesirable sampling error but keeping the sequence effect that may be desirable. The values of the descriptive sample do not vary; only their sequences will vary between different simulation runs. In spite of its advantages, this method is known to have two problems: it can be biased and its strict operation requires a prior knowledge of the sample size [25].

2.4 Refined Descriptive Sampling

RDS is concerned with a block that must be situated inside a generator aiming to distribute regular samples of prime number sizes $p_1, p_2, \ldots$ when required by the simulation. We stop the process when the simulation terminates.

In this procedure, each run is determined by a block of different prime numbers. Contrary to DS, the values of refined descriptive sample vary between different runs since the sample size is a prime number randomly selected. The generation process of the sample values is deterministic whereas the generation of the prime numbers as well as the sequence of the sample values are random.

2.5 Comparison

Several empirical comparisons on a PERT network, an M/M/1 queue, an inventory system and Newsboy problem [29] show that the estimates of the output random variables parameters produced through simulation using DS are with lower variance than those obtained by RS. In [30] there is a discussion suggesting that DS has a lower variance than LHS whereas Saliby and Pacheco [31] compared the efficiency of six Monte Carlo simulation sampling methods, namely QMC using Halton, Sobol and Faure numeric sequences, DS, LHS and RS in two finance applications: a project risk analysis and a correlated stock portfolio where it was shown that DS and LHS produced the best results. Tari and Dahmani [36] have shown that RDS is an efficient sampling in simulation studies developing methods. This reference was concerned with ensuring that the implementation of RDS method is safe, correctly designed and fitted to any simulation in an economical and undemanding manner. The authors compare the efficiency of DS and RDS on a flow shop system [37] and a production system [38] showing that RDS produces better results than DS and RS.

However, the aim of any sampling procedure is to represent truly the population distribution from which it is generated so that any simulation estimates drawn from it can be safely implemented in the real system. Therefore, RDS outperform all other sampling methods: DS, RS, LHS and QMC methods and it is capable of substantially reducing the cost of running simulation experiments.

3 THE USE OF REFINED DESCRIPTIVE SAMPLING

Without loss of generality, we assume that one input random variable X drives the simulation and one output random variable with k parameters θ_j to be estimated is observed through simulation.

3.1 The Refined Descriptive Samples

Suppose that m prime numbers have been used in a simulation run. Formally, in RDS, regular sample values are generated for the input random variable X as required by the simulation using the inverse transform method such as

$$(xd)^i_{i+\sum_{i=1}^{q-1} p_i} = H^{-1}\left(r^i_{i+\sum_{i=1}^{q-1} p_i}\right) \text{ for } i = 1, 2, \ldots, p_q \text{ and } q = 1, 2, \ldots, m$$

where

$$\left\{r^1_{1+\sum_{i=1}^{q-1} p_i}, r^2_{2+\sum_{i=1}^{q-1} p_i}, \ldots, r^{p_q}_{\sum_{i=1}^{q} p_i}\right\} \text{ for } q = 1, 2, \ldots, m$$

are considered as subsets of dependent regular numbers of prime size $p_1, p_2, \ldots, p_m$ uniformly distributed between $[0, 1]$ and are obtained by the following formula

$$r^i_{i+\sum_{i=1}^{q-1} p_i} = \frac{i - 0.5}{p_q} \qquad \text{for } i = 1, 2, \ldots, p_q \text{ and } q = 1, 2, \ldots, m,$$

and $H^{-1}$is the inverse cumulative function of the input random variable X.

The refined descriptive samples of prime number size p_q are obtained by:

1. generating the subsets of regular numbers

$$\left(r^1_{1+\sum_{i=1}^{q-1} p_i}, r^2_{2+\sum_{i=1}^{q-1} p_i}, \ldots, r^{p_q}_{\sum_{i=1}^{q} p_i}\right)$$

   of size a randomly chosen prime number p_q, $q = 1, 2, \ldots, m$

2. randomizing their sequence for any p_q

3. computing the regular sample values of the input random variable $xd^{i}_{i+\sum_{i=1}^{q-1} p_i}$ for $i = 1, 2, \dots, p_q$ as required by the simulation.

3.2 RDS Simulation Estimates

In a given run, the use of RDS procedure leads to the following m estimates of the parameters θ_j, $j = 1, 2, \dots, k$

$$\begin{aligned}
(Yr)^1_j &= F_j\left(r^1_1, r^2_2, \dots, r^{p_1}_{p_1}\right) \\
&\qquad \text{for } j = 1, 2, \dots, k \\
(Yr)^2_j &= F_j\left(r^1_{1+p_1}, r^2_{2+p_1}, \dots, r^{p_2}_{p_2+p_1}\right) \\
&\qquad \text{for } j = 1, 2, \dots, k \\
&\vdots \\
(Yr)^m_j &= F_j\left(r^1_{1+\sum_{i=1}^{m-1} p_i}, r^2_{2+\sum_{i=1}^{m-1} p_i}, \dots, r^{p_m}_{\sum_{i=1}^{m} p_i}\right) \\
&\qquad \text{for } j = 1, 2, \dots, k
\end{aligned}$$

such as

$$\sum_{i=1}^{m} p_i \geq n$$

and conventionally

$$\sum_{i=1}^{0} p_i = 0$$

where F_j, $j = 1, 2, \dots, k$ is a simulation function usually defined by a program that relates the input variables with each estimator, and $\left(r^1_{1+\sum_{i=1}^{q-1} p_i}, r^2_{2+\sum_{i=1}^{q-1} p_i}, \dots, r^{p_q}_{\sum_{i=1}^{q} p_i}\right)$, $q = 1, 2, \dots, m$ are the subsets of regular numbers uniformly distributed over the range 0 and 1.

Therefore, in a given run, the use of RDS method leads to the following sampling estimates of θ_j, $j = 1, 2, \dots, k$, defined by the average of these estimates

$$(Yr)_j = \frac{1}{m}\sum_{i=1}^{m} (Yr)^i_j \qquad \text{for } j = 1, 2, \dots, k.$$

Remark 1. If N replicated runs are needed to run a simulation experiment, it is then necessary to consider N blocks of m_1, $m_2, \dots, m_N$ regular samples.

4 PARALLELISM

There are several parallel programming models developed and two main variants of parallel programming models classified are shared and distributed memory based parallel programming models. The parallel programming model is often influenced by the type of the application. Then, a better programming model is needed that facilitates easy development and on the other hand porting high performance.

A programmed application using threads is more appropriate on shared memory because the threads share the same memory space. On the other hand, if we have a large amount of data which cannot take place in the memory of one processor, it is more appropriate to distribute the data on different processors. In this case, to ensure a better data distribution between different nodes, the distributed memory based parallel programming models is the most appropriate. Nevertheless, both architectures can be appropriate if the program is processed with small amounts of data.

We can find various classifications for parallel architectures in the literature. Some examples are the classification of Flynn [10], Taxonomy of Skililcorn [32] and the classification of Duncan [7]. They are all based on particular and important points of the architecture but the classification of Flynn is the most popular one.

5 THE PROPOSED PARALLEL SIMULATION

In this section, we propose an optimal parallel Monte Carlo simulation algorithm using refined descriptive sampling which is implemented on parallel architecture computers.

Let us first define the parallel run time of a program by the following formula

$$T_{parallel} = \text{Computation } time + \text{Communication } time$$

where the computation time is a locally processing time and the communication time is given by

$$\text{Communication } time = \text{Latency } time + \text{Over-cost } time + \text{ Transferred } time$$

where

- latency time is the initialization time of the network parameters,
- over-cost time is the make-ready time of the message,
- transferred time is the necessary time for transferring the message.

Suppose that $Q+1$ processors are taking part in the simulation experiment. We appoint one processor as the master called M and the remaining Q by the slaves' processors or workers called W_1, W_2, …, W_Q.

It is well known that a program composed by more than one independent tasks can be able to be run simultaneously on multiple processors in order to obtain results faster [27]. Usually, in parallelism using SPMD model, the program distribution is done in turn between the various processors using "Round Robin" discipline and if a MC simulation experiment is carried out in parallel with N replicated runs then the number of messages sent by the master M for all workers W_i, $i = 1, 2, \ldots, Q$ is equal to N.

The proposed parallel MC simulation consists of execution of the same program over all workers with different data given by RDS generator which are independent from one processor to another. Furthermore, to ensure the independence of the generated prime numbers related to RDS sampling method itself, we suppose independent prime numbers generators from one processor to another. To reduce the transferred time, the master distributes all inputs of the program and the number $\frac{N}{Q}$ of runs to be replicated for each slave processor at once such as each processor carries out the same copy of such program $\frac{N}{Q}$ times (where N is a multiple of Q to ensure that $\frac{N}{Q}$ is an integer). The reception of partial simulation results by the master is done in the same manner as the distribution. In the following we show how we reduced the transferred time. Let T_i to be the communication time between the master M and each worker W_i, $i = 1, 2, \dots Q$ where T_i is the same value for all exchanges, say T. Then the transferred time of distribution from M to Q is $T \times Q$, and the transferred time of reception from Q to M is also $T \times Q$, then the transferred time is $2 \times T \times Q$. Taking T as a unit time, the number of messages is then reduced from $2N$ to $2Q$ (distribution and reception). Then the influence of the communication time is highly controlled and therefore it is regarded as an insignificant time compared to the computation time. Then,

$$T_{parallel} \approx \text{computation time.}$$

Note that at the end of reception of all partial results, the master computes the performance measures of the simulation.

It is well known that load balancing is one of the most significant concepts in parallel computing. In the proposed parallel MC simulation algorithm, we passed around the load balancing because the considered application is regular (each worker executes the same program and the same number of runs).

In the proposed parallel program, according to the replication of simulation runs and their independence, SPMD model has been selected for a better programming. The sequential part of the proposed parallel program denoted by α is processed just at the beginning and at the end of the program. The beginning of the program by reading input data and their distribution over all possible workers and the end of the program by computing all performance measures using the partial results obtained when all slaves' processors have finished running their replicated runs. All these instructions are done by the master. However, the speedup of processing by a parallel system with $Q + 1$ processors is formulated by Gustafson's law [12] given by

$$S_{Q+1} = (Q + 1) - Q\alpha.$$

If the part of the code executed by the workers is large, then α can be neglected, therefore $S_{Q+1} \approx Q + 1$.

6 RDS ALGORITHM AND ITS IMPLEMENTATION

Before each simulation run, we generate randomly a prime number p. To do so, we first generate a random number and then we test if it is a prime number using an improvement of the test of primality, the so-called prime number generator. A value of the sample is then generated to be used in the simulation experiment. We stop generating sample values when the simulation terminates.

6.1 RDS Algorithm

- a) Initialization for the experiment.
  - a_1) Before each simulation run, generate a sequence of distinct prime numbers.
  - a_2) Choose a prime number p from this sequence randomly without replacement.
  - a_3) Generate the subset of regular numbers r_i, $i = 1, 2, \dots, p$ and store them in an array R.
- b) Initialization for the sub-run. At the beginning of every sub-run, let $ip := 1$.
- c) Sampling without replacement during the sub-run:
  - c_1) if $ip > p$ then go to (d)
  - c_2) randomly generate an integer $iaux \in [ip, p]$
  - c_3) interchange $r(ip)$ with $r(iaux)$
  - c_4) generate one observation xd_i.
    If a descriptive sub-set value is not required, stop and collect the final results from the last prime number used and go to (e)
  - c_5) otherwise let $ip := ip + 1$ and go to (c_1).
- d) Collect the results after each sub-run and go to (a_2).
- e) Collect the results after each run.

6.2 Random Number Generation

The mixed congruential generator of C++ is represented by the function rand() and obtained by the following recurrence relation

$$x_{n+1} = ax_n + c \bmod m$$

where

- $m > 0$ is a modulus,
- a and c, $0 \leq a < m$ and $0 \leq c < m$ are positive integers called the multiplier and the increment, respectively
- and x_0, $0 \leq x_0 < m$ is the initial value of the generating sequence.

The pseudo random number generation function rand() is given by the following algorithm:

```
#Let us define RAND_MAX 32 767/($2^{15} - 1$)
// as the greatest value that may be generated
Begin
    SEED = ($1\,103\,515\,245 \times$ SEED + $12\,345$) mod $2^{15}$
    X = SEED
    Returns integer in range [0, RAND_MAX]
End // By repeating this process, we get a series of random numbers between 0
and $2^{15} - 1$ reproducible since the initial value of SEED is always the same but if
different sequences are needed, we change the function rand() by srand().
```

6.3 Test of Primality

```
// A function of test of primality for a given $x$ number
Begin
    Is_Prime($x$)
    {// $x$ is an unspecified generated random number
    // Lower than 2: not prime
    If ($x < 2$) Return (False);
    // Equal 2: prime number
    If ($x == 2$) Return (True);
    // even number else than 2: not prime
    If (($x \% 2$) == 0) Return (False);
    // Test all $i$ divisor between 3 and $\left[x^{1/2}\right]$
    $i = 3$;
    While ($i * i <= x$)
    {
        // If this number is divisible by $i$: It is not prime
        If (($x \% i$) == 0) Returns (False);
        // go to the next divisor
        $i += 2$;
    };
    // Any i number between 3 and $\left[x^{1/2}\right]$ would split $x$:
    It is prime
    Return (True);
    };
End
```

6.4 Prime Number Generator

We first generate an unspecified integer number, then we multiply it by two and add one in order to make the generated number an odd number since all prime numbers are odd except number 2 which is ignored; finally, we test if the odd number is a prime number by successive division on odd numbers, starting from three with a step of two. If it is a prime number we use it in running simulation by generating observations of input random variables, otherwise we go back to generate another unspecified integer. In this manner, we reduce the time allocated to the generation of prime numbers by 75 %; we then reduce the time of sequential running simulation experiments and in this way, we again reduce the parallel time by reducing the computation time.

```
// A modified function of test of primality for an unspecified $x$ an odd number
// Let $y$ to be an integer randomly generated between $[0, 2^{15} - 1]$
Begin
    x=2*y+1;
    IsPrime(x)
    {
        // Test all i odd divisor between the integer 3 and $[x^{1/2}]$
        i = 3;
        While (i*i <= x)
        {
            // If this number x is divisible by i: It is not prime
            If ((x % i) == 0) Returns false;
            // Go to next divisor
            i += 2;
        };
        // Any i number between 3 and $\left[x^{1/2}\right]$ would split $x$: It is then prime
        Returns (True);
    };
End
// If the function IsPrime(x) return false, another random integer $y$ is again generated until a prime number is obtained.
```

6.5 Regular Numbers Generation

```
// Let $p$ to be a prime number generated by the above algorithm
// Let $R$ to be the array containing the regular numbers $r_i$, $i = 1, 2, \ldots, p$
For $i$ := 1 to $p$ do
Begin
    $r_i = \frac{i-0.5}{p}$
End for
// Initialization for the sub-run. At the beginning of every sub-run, let $ip$ := 1, then it increments at each iteration $ip$ := +1.
// Let $k$ to be an intermediate variable
// Randomly generate an integer iaux ∈ [$ip$, $p$] ($ip$ route index of vector R)
Begin (permutation)
```

```
    $k := r(ip)$
    $r(ip) := r(iaux)$.
    $r(iaux) := k$
End (permutation)
```

6.6 Refined Descriptive Samples Generation

Generation of an observation x_i of an input random variable X.
// Let $x_i = F^{-1}(r_i)$ where F^{-1} is the inverse cumulative function of X.
// In the first problem, the negative exponential distribution of parameter λ is used.
// The observations of this distribution are given by

$$x_i = -\frac{1}{\lambda} \ln (r(ip))$$

// In the second problem, a discrete uniform distribution is used.
// In the third problem, an empirical distribution is given.
// The observations of both distributions are generated using top hat method.

7 SAMPLE PROBLEMS

In this section, we evaluate parallel Monte Carlo simulation, performance measures of a stable M/M/1 queuing system, a Pert network and the Newsboy problem. In each sample problem, we choose the same input parameters for all carried out experiments.

We summarize each experiment by computing the mean and variance of the estimates of the output random variables parameters with both sampling methods.

7.1 The M/M/1 Queuing Problem

This problem concerns the steady state of queue size and queue waiting time distributions for an M/M/1 queue. In this problem, there are two input random variables, the inter-arrival time and the service time following both an exponential distribution of parameter respectively λ and μ, and two response variables are observed through simulation. The queue size and the queue waiting time have both one parameter, the mean, Lq and Wq to be estimated by $\overline{size}$ and $\overline{Time}$, respectively.

7.1.1 Data Structure

For each experiment, we define a record with the following structure:

- N: integer number
- λ, μ, T: float numbers.

For each input random variable, we define a record with the following structure:

- p: a prime number defining the size of the regular numbers subset;
- R: array $[1 \ldots p]$ of real numbers containing the subset of regular numbers;
- ip: integer pointing to the first available r element to be drawn. If $ip = 1$, no element has been drawn yet. If $ip > p$, a full subset of regular numbers has already been drawn.

7.1.2 The Parallel Algorithm

In this sub-section, we define the main steps of the proposed and implemented parallel algorithm.

```
Begin
 1. The Master reads the simulator parameters
        $N$ : Number of replicated runs
        $\lambda$ : Inter-arrival rate
        $\mu$ : Service rate
        $T$ : Simulation period
 2. The Master distributes the parameters to each slave processor
        For $k$ from 0 to $Q-1$
            send $(\frac{N}{Q}, \lambda, \mu, T)$ to processor $k$
 3. Each slave processor carries out $\frac{N}{Q}$ simulation runs of the sample problem using
    RDS algorithm given in sub-section 6.1 to generate input refined descriptive
    samples
 4. Gathering together the simulation results obtained on each slave processor
        For $k$ from 0 to $Q-1$ send all computed estimates of the $\frac{N}{Q}$ runs
        to the Master
 5. The Master computes the overall mean of each studied parameter based on all
    runs
 6. Display the final results by the Master
end
```

7.1.3 Empirical Results

For both carried out experiments, the number of replicated runs is taken to be equal to the number of available slave processors. The simulation period was set equal to 1 200 units; we take $\lambda = \frac{1}{3}$ and $\mu = \frac{1}{2}$. For these problem parameters, the theoretical values of the output variables parameters under study are $Lq = 1.33\,\text{mn}$ and $Wq = 4\,\text{mn}$. In the first experiment, we take $Q = N = 2$ and in the second experiment we take $Q = N = 5$. The observed results are listed in Table 1.

We can see from Table 1 that when the number of replicated runs increases, the observed results get closer to the theoretical values.

	$Q = N = 2$		$Q = N = 5$	
Estimate	Mean	Var	Mean	Var
$\overline{size}$	2.06	0.11	1.56	0.08
$\overline{Time}$	4.43	0.09	4.20	0.05

Table 1. Empirical results showing the parallelization efficiency for both experiments

7.2 The Pert Network

7.2.1 Description of the Problem

This problem concerns a simple Pert network already studied by Kleindorfer [16]. As shown in Figure 1, this network has eight activities. All activities durations are independent and identically distributed random variables following a discrete uniform distribution defined by

$$f(d) = 0.2 \text{ for } d = 1, \ldots, 5$$

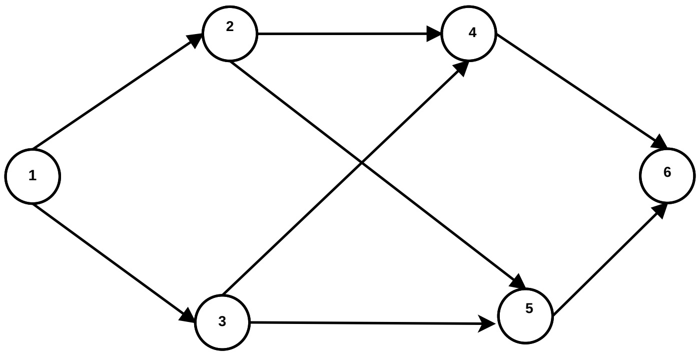

Fig. 1. The simulated Pert network

In this problem, there is one response variable with two parameters – the mean and the standard deviation – to be estimated. The observed response variable is the total project duration. Its parameters are both estimated by $\overline{DT}$ and S_{DT}.

7.2.2 Empirical Results

Four simulation experiments of different replicated runs were carried out on the studied problem, where each run was defined by 50 observations of the total project duration. We take $Q = 5$ in each experiment and for different number of runs, the observed results are given in Table 2 below.

We can see from Table 2 that the growing up of the number of replicated runs does not affect the observed results.

Estimate\N	50		200		250		300	
	Mean	var	Mean	var	Mean	var	Mean	var
$\overline{DT}$	10.13	0.36	10.13	0.41	10.11	0.42	10.13	0.41
S_{DT}	2.24	1.05	2.30	0.92	2.3	0.95	2.29	0.95

Table 2. Empirical results showing the parallelization efficiency for different experiments

7.3 The Newsboy Problem

7.3.1 Description of the Problem

This problem concerns a simple inventory problem already studied by Kaufmann and Faure [15]. Daily, a newsboy buys B issues of a newspaper at 0.5 £ each. The selling price is 1 £. At the end of the day, all remaining issues are restored and the salvage price for surplus newspapers is 0.2 £ each. Daily demand (D) is independent and identically distributed according to the following statistical distribution where Freq stands for frequency and C freq stands for cumulative frequency.

D	Freq	C freq	D	Freq	C freq	D	Freq	C freq	D	Freq	C freq
0	0	0	13	1	17	26	4	64	39	1	94
1	0	0	14	3	20	27	3	67	40	2	96
2	1	1	15	3	23	28	3	70	41	0	96
3	1	2	16	3	26	29	4	74	42	1	97
4	1	3	17	4	30	30	2	76	43	1	98
5	2	5	18	3	33	31	3	79	44	0	98
6	1	6	19	4	37	32	3	82	45	0	98
7	1	7	20	3	40	33	2	84	46	1	99
8	1	8	21	4	44	34	2	86	47	0	99
9	2	10	22	5	49	35	2	88	48	0	99
10	2	12	23	4	53	36	1	89	49	1	100
11	1	13	24	4	57	37	2	91	50	0	100
12	3	16	25	3	60	38	2	93	> 50	0	100

Table 3. Cumulative frequency of daily demand

The newsboy wishes to know the quantity B of newspapers to purchase in order to maximize its daily profit. Then, the simulation purpose is to study the daily profit distribution P varying the quantity to be bought until 50 and then searching the optimal profit value and the corresponding quantity to be purchased. In this problem, there is one response variable with the mean parameter to be estimated by $\overline{P}$.

7.3.2 Implemented Algorithm

```
Begin
For 1, $B = 1$ until 50 do
```

```
    For 2, $j = 1$ until 100 do
    Simulate daily demand D with RDS algorithm and compute the expected profit,
    If $B < D$ then expected profit $= B \times 0.5$
    otherwise expected profit $= D \times 0.5 - (B - D) \times 0.3$
    end if
end for 2
    Save the daily expected profit value
    Compute the mean of the daily expected profit over 100 working days
    Save the expected profit mean value
end for 1
For 3, $i = 1$ until 50
    Look for the optimal expected profit
    Look for the corresponding quantity $B$ of newspapers to purchase
end for 3
Display the optimal profit and the optimal quantity to purchase
end
```

7.3.3 Empirical Results

Seven simulation experiments of different replicated runs were carried out on the newsboy problem, where each run was defined by 100 days for each bought quantity. We simulate the daily demand distribution according to Table 3 and the newsboy problem according to the above algorithm. The summarized results are given below.

	N	10	20	40	70	100	200	350
optimal	$Mean\ (\overline{P})$	8.51	8.01	7.91	8.06	7.75	7.89	7.53
	B	24	22	23	26	23	22	21
	$Var\ (\overline{P})$	0.11	0.28	0.49	0.68	0.91	0.5	0.53

Table 4. Empirical results showing the parallelization efficiency for different experiment

We can see from the simulation results of Table 4 that the minimum variance is obtained within just 10 replicated runs. We then suggest to the newsboy to purchase a quantity of newspaper equal to 24 in order to make a profit of 8.51 £.

8 CONCLUSIONS AND REMARKS

The proposed MC parallelization using the best sampling RDS procedure reduces the computation time when the number of runs and/or the duration of simulation are large enough, while preserving the assets of the sampling method, its efficiency and safety. Since this method works by replication, then it was naturally parallelized.

Providing criteria for assessment is essential, we define three main criteria of optimization such as the parallel runtime, the number of processors (scalability) and the cost. The parallel runtime is the time that elapses from the moment a parallel computation starts to the moment the last processing element finishes execution. For a given number of runs, whatever the number of processors is, the simulation results obtained in parallel execution are similar to those obtained sequentially, but the parallel runtime of simulation experiments (program) is different and inversely proportional to the number of available processors according to the following formula

$$T_{parallel} \approx \frac{T_{\ sequential}}{Q}$$

while the proportion of computation for a slave processor is very large compared to the communication time between the master and each slave processor.

On the other hand, increasing the number of processors is beneficial (rentable) for the proposed parallelization just in the case where the computation time of any added worker is significantly larger than the communication time and in this case the scalability of the SPMD model is preserved.

Regarding cost, it is considered as the communication time and the cost of idle machines. In this case, the cost is controlled because the communication time is insignificant as mentionned in Section 5. Given that all workers have the same charge, we have then passed around the idle machines. Accordingly, the proposed parallelization is optimal.

A stable M/M/1 queueing system, a Pert network and the Newsboy problem were simulated by using the proposed parallel algorithm, but any other application using replications can be solved by this algorithm.

REFERENCES

[1] Ahn, J. S.—Danzig, P.: Packet Network Simulation: Speedup and Accuracy Versus Timing Granularity. IEEE/ACM Transactions on Networking, Vol. 4-5, 1996, pp. 743–757.

[2] Alme, H. J.—Rodrigue, G.—Zimmerman, G.: Domain Decomposition for Parallel Laser-Tissue Models With Monte Carlo Transport. In H. Niederreiter and J. Spanier: Monte Carlo and Quasi Monte Carlo Methods (Springer, 1998), pp. 86–97.

[3] Avramidis, A. N.—L'Ecuyer, P.: Efficient Monte Carlo and Quasi-Monte Carlo Option Pricing Under the Variance-Gamma Model. Management Science, Vol. 52, 2006, No. 12, pp. 1930–1944.

[4] Butler, R. M.—Lusk, E. L.: Monitors, Message and Clusters: The p4 Parallel Programming System. Parallel Computing, Vol. 20, 1994, No. 4, pp. 547–564.

[5] Booth, T. E.: Adaptively Learning an Importance Function Using Transport Constrained Monte Carlo. In: H. Niederreiter and J. Spanier: Monte Carlo and Quasi Monte Carlo Methods (Springer, 1998), pp. 1–15.

[6] Chergui, J.—Lavallée, P. F.: OpenMP Multi Tasks Parallelism for Shared Memory Machines. In French, Institut du developpement et des ressources en Informatique scientifique, 2006.

[7] Duncan, R.: A Survey of Parallel Computer Architectures, Computer, Vol. 23, 1990, No. 2, pp. 5–16.

[8] Doucet, A.—de Freitas, N.—Gordon, N.: Sequential Monte Carlo methods in Practice. Springer Verlag, 2001.

[9] Fishman, G. S.: Monte-Carlo: Concepts, Algorithms and Applications. Springer-Verlag, 1997.

[10] Flynn, M. J.: Very High Speed Computing Systems. In proceeding IEEE Vol. 54, 1966, No. 12, pp. 1901–1909.

[11] MPI Forum, MPI: A Message Passing Interface Standard. 2003, `http://www.mpi-forum.org/docs/mpi1-report.pdf`.

[12] Gustafson, J. L.: Reevaluating Amdahl's Law. CACM, Vol. 31, 1988, No. 5, pp. 532–533.

[13] Gest, A.—Beguelin, A.—Dongarra, J.—Jiang, W.—Manchek, R.—Sunderam, V.: PVM: Parallel Virtual Machine, a User Guide and Tutorial for Network. Parallel Computing. MIT Press, 1994.

[14] Hoshino, N.—Takemura, A.: On Reduction of Finite Sample Variance by Extended Latin Hypercube Sampling. Bernoulli, Vol. 6, 2000, No. 6, pp. 1035–1050.

[15] Kaufmann, A.—Faure, R.: Invitation to Operational Research. In French, Bordas, Paris, Dunod entreprise, 1975.

[16] Kleindorfer, G. B.: Bounding Distributions for a Stochastic Acyclic Network. Journal of the Operational Research Society, Vol. 19, 1971, pp. 1586–1601.

[17] L'Ecuyer, P.: Quasi Monte Carlo Methods for Simulation. Winter Simulation Conference. In Chick, S., Sanchez, P. J., Ferrin, D., Morrice, D. J. ed., 2003.

[18] Loh, W. L.: On Latin Hypercube Sampling. The annals of statistics, Vol. 24, 1996, pp. 2058–2080.

[19] McKay, M. D.—Beckman, R. J.—Conover, W. J.: A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. Technometrics. Vol. 21, 1979, pp. 239–245.

[20] Niederreiter, H.: Random Number Generation and Quasi Monte-Carlo Methods. Philadelphia: CBMS-SIAM, 63, 1992.

[21] Okten, G.: A Probabilistic Result on the Discrepancy of a Hybrid-Monte Carlo Sequence and Applications. Monte Carlo Methods and Applications, Vol. 2, 1996, No. 4, pp. 255–270.

[22] Okten, G.—Tuffin, B.—Bugaro, V.: A Central Limit Theorem and Improved Error Bounds for a Hybrid-Monte Carlo Sequence With Applications in Computational Finance. Journal of Complexity, Vol. 22, 2006, No. 4, pp. 435–458.

[23] Owen, A. B.: Monte Carlo Variance of Scrambled Net Quadrature. SIAM J. Numer. Anal., Vol. 34, 1997, No. 5, pp. 1884–1910.

[24] Owen, A. B.: Monte Carlo, Quasi Monte Carlo, and randomized Quasi Monte Carlo. In H. Niedrreiter and J. Spanier: Monte Carlo and Quasi Monte Carlo Methods, Springer, 1998, pp. 86–97.

[25] Pidd, M.: Computer Simulation in Management Science. 4th Edition. John Wiley and Sons, Chichester, 1998.

[26] Robert, C. P.—Casella, G.: Monte Carlo Statistical methods. 2nd Edition. Springer Science, Business Media Inc., 2004.

[27] Rodgers, D. P.: Improvements in Multiprocessor System Design. ACM SIGARCH Computer Architecture News archive, New York, ACM 13, Vol. 3, 1985, pp. 225–231.

[28] Ross, S. M.: Simulation. 2nd Edition. Academic Press, 1997.

[29] Saliby, E.: Descriptive Sampling: A Better Approach to Monte Carlo Simulation. Journal of the Operational Research Society, Vol. 41, 1990, No. 12, pp. 1133–1142.

[30] Saliby, E.: Descriptive Sampling: an Improvement over Latin Hypercube Sampling. Winter Simulation Conference, 1997, pp. 230–233.

[31] Saliby, E.—Pacheco, F.: An Empirical Evaluation of Sampling Methods in Risk Analysis Simulation: Quasi Monte Carlo, Descriptive Sampling, and Latin Hypercube Sampling. Winter Simulation Conference, 2002, pp. 1606–1610.

[32] Skililcorn, D. B.: A Taxonomy for Computer Architectures, IEEE Computer 21, Vol. 11, 1985, pp. 46–57.

[33] Sloan, I. H.—Joe, S.: Lattice Methods for Multiple Integration. Oxford: Clarendon Press, 1994.

[34] Tari, M.: Ourbih, Improvement of Descriptive Sampling: Application to a Production and a Scheduling Workshop Problems. In French. Doctorat en Science, Bejaia University, Algeria.

[35] Tari, M.—Dahmani, A.: The Refining of Descriptive Sampling. International Journal of Applied Mathematics and Statistics (IJAMAS), Vol. 3, 2005, pp. 41–68.

[36] Tari, M.—Dahmani, A.: Refined Descriptive Sampling: A Better Approach to Monte Carlo Simulation. Simulation Modelling Practice and Theory, Vol. 14, 2006, pp. 143–160.

[37] Tari, M.—Dahmani, A.: Flowshop Simulator Using Different Sampling Methods. Operational Research: An International Journal (ORIJ), Vol. 5, 2005, No. 2, pp. 261–272.

[38] Tari, M.—Dahmani, A.: The Three Phase Discrete Event Simulation Using Some Sampling Methods. International Journal of Applied Mathematics and Statistics (IJAMAS), Vol. 3, 2005, pp. 37–48.

[39] Taylor, S. J. E.—Hlupic, V.—Robinson, S.—Ladbrook, J.: GroupSim: Investigating Issues in Collaborative Simulation Modelling. Operational Reaserch Society Simulation Workshop 2002, pp. 11–18, 2002.

[40] Tuffin, B.—Le Ny, L. M.: Parallelization of a Combination of Monte Carlo and Quasi-Monte Carlo Methods Networks and Application on Queuing Networks. In French, RAIRO Operations Research 34, 2000, pp. 85–98.

Abdelouhab Aloui is the Assistant Head of Computer Science Department, a member of the Scientific Committee of the Faculty of Sciences. He is a member of the Research Team "Simulation & Computer Science" at Applied Mathematics Laboratory. He received his Engineer Degree at Mouloud MAMMERI University (Tizi ouzou, Algeria), and his Magister's Degree at the University of Abderrahmane Mira (Bejaia, Algeria) both in Computer Science. He is currently a Doctorate student at the Department of Computer Science, Abderrahmane Mira University. He is a member of an Algerian National Research Project (PNR) and CNEPRU Project. He is also the author of several scientific journal papers and conference contributions, and a member of the organizing committee at international scientific conferences. He is working as a lecturer at the above-mentioned department in Bejaia. His research interests are in refined descriptive sampling, Monte Carlo methods and parallel programming.

Megdouda Ourbih Tari is the Chairman of the Scientific Committee of the Department of Mathematics, the Head of the License of "Statistics and data processing" and the Head of the Master of "Statistics and Decision Analysis" at the Department in Bejaia University. She is also the Head of Research Team "Simulation & Computer Science" at Applied Mathematics Laboratory. She received MPhil degree from Lancaster University (England), Doctorate degree at Bejaia University (Algeria), both in Operational Research and HDR. She is Algerian National Research Project (PNR) and CNEPRU Project Manager. She is the author of numerous scientific journal papers, contributions, and a member of the organizing and scientific committees at international scientific conferences. She also lectures at Bejaia University and acts as supervisor and consultant for Doctoral, Magister, Master and License studies.