A Strategy for Parallel Implementations of Stochastic Lagrangian Simulation

Lionel Lenôtre

A Strategy for Parallel Implementations of Stochastic Lagrangian Simulation

Lionel Lenôtre

L. Lenôtre
Project-Team Sage
Inria Rennes Bretagne-Atlantique
Campus de Beaulieu 35042 Rennes Cedex France
e-mail: lionel.lenotre@inria.fr

Abstract: In this paper, we present some investigations on the parallelization of stochastic Lagrangian simulations. The challenge is the proper management of the random numbers. We review two different object-oriented strategies: to draw the random numbers on the fly within each MPI's process or to use a different random number generator for each simulated path. We show the benefits of the second technique which is implemented in the PALMTREE software developed by the Project-team Sage of Inria Rennes. The efficiency of PALMTREE is demonstrated on two classical examples.

1. Introduction

Monte Carlo simulation is a very convenient method to solve problems arising in physics like the advection-diffusion equation with a Dirichlet boundary condition

$$\begin{cases} \dfrac{\partial}{\partial t}\, c(x,t) = \mathrm{div}(\sigma(x)\cdot\nabla c(x,t)) - v(x)\nabla c(x,t)), \quad \forall (x,t)\in \overline{D}\times[0,T], \\ c(x,0) = c_0(x), \quad \forall x \in \overline{D}, \\ c(x,t) = 0, \quad \forall t\in[0,T] \text{ and }, \ \forall x\in\partial D, \end{cases} \tag{1}$$

where, for each $x \in D$, $\sigma(x)$ is a d-dimensional square matrix which is definite, positive, symmetric, $v(x)$ is a d-dimensional vector such that $\mathrm{div}(v(x)) = 0$, $D \subset \mathbb{R}^d$ is a regular open bounded subset and T is a positive real number. In order to have a well-posed problem [4, 5] and to be able to use later the theory of stochastic differential equations, we required that σ satisfies an ellipticity condition and has its coefficients at least in $\mathcal{C}^2(\overline{D})$, and that v is bounded and in $\mathcal{C}^1(\overline{D})$.

Interesting computations involving the solution $c(t,x)$ are the moments

$$M_k(T) = \int_D x^k\, c(T,x)\, dx, \quad \forall k \geq 1 \text{ such that } M_k(T) < +\infty.$$

One possibility for their computation is to perform a numerical integration of an approximated solution of (1). Eulerian methods (like Finite Difference Method, Finite Volume Method or Finite Element Method) are classical to obtain such an approximated solution. However, for advection-diffusion problems, they can induce numerical artifacts such as oscillations or artificial diffusion. This mainly occurs when advection dominates [7].

An alternative is to use Monte Carlo simulation [6, 19] which is really simple. Indeed, the theory of stochastic processes implies that there exists $X = (X_t)_{t\geq 0}$ whose law is linked to (1) and is such that

$$M_k(T) = \mathbb{E}[X_T^k]. \tag{2}$$

The above expectation is nothing more than an average of the positions at time T of particles that move according to a scheme associated with the process X. This requires a large number of these particles to be computed. For linear equations, the particles do not interact with each other and move according to a Markovian process.

The great advantage of the Monte-Carlo method is that its rate of convergence is not affected by the curse of dimensionality. Nevertheless, the slowness of the rate caused by the Central-Limit theorem can be considered as a drawback. Precisely, the computation of the moments requires a large amount of particles to achieve a reliable approximation. Thus, the use of supercomputers and parallel architectures becomes a key ingredient to obtain reasonable computational times. However, the main difficulty when one deals with parallel architectures is to manage the random numbers such that the particles are not correlated, otherwise a bias in the approximation of the moments is obtained.

In this paper, we investigate the parallelization of the Monte Carlo method for the computation of (2). We will consider two implementation's strategies where the total number of particles is divided into batches distributed over the Floating Point Units (FPUs):

1. **SAF**: the Strategy of Attachment to the (FPUs) where each FPU received a Virtual Random Number Generator (VRNG) which is either different independent Random Number Generators (RNGs) or copies of the same RNG in different states [10]. In this strategy, the random numbers are generated on demand and do not bear any attachment to the particles.
2. **SAO**: the Strategy of Attachment to the Object where the particles carries their own Virtual Random Number Generator.

Both schemes clearly carry the non correlation of the particles assuming that all the drawn random numbers have enough independence which is a matter of RNGs.

Sometimes particles with a singular behavior are encountered and the examination of the full paths of such particles is necessary. With the SAF, a particle replay requires either to re-run the simulation with a condition to record only the positions of this particle or to keep track of the random numbers used for this particle. In both cases, it would drastically increase the computational time and add unnecessary complications to the code. On the contrary, a particle replay is straightforward with the SAO.

The present paper is organized in two sections. The first one describes SAF and SAO. It also treat of the work done in PALMTREE, a library we developed with the generator RNGStreams [11] and which contains an implementation of the SAO. The second section presents two numerical experiments which illustrate the performance of PALMTREE [17] and the SAO. Characteristic curves like speedup and efficiency are provided for both experiment.

2. Parallel and Object-Oriented Implementations in Monte Carlo

All along this section, we assume that we are able to simulate the transition law of particles undergoing a Markovian dynamics such that there is no interactions between them. As a result, the

presentation below can be applied to various Monte Carlo schemes involving particle tracking where the goal is to compute moments. Moreover, this shows the flexibility of the implementation we choose.

2.1. An Object-Oriented Design for Monte Carlo

C++ offers very interesting features which are of great help for a fast execution or to treat multidimensional processes. In addition, a consistent implementation of MPI is available in this language. As a result, it becomes a natural choice for PALMTREE.

In what follows, we describe and motivate the choices we made in the implementation of PALMTREE. We refer to a FPU as a MPI's process.

We choose to design an object called the Launcher which conducts the Monte Carlo simulation. Roughly speaking, it collects all the generic parameters for the simulation (the number of particles or the repository for the writing of outputs). It also determines the architecture of the computer (cartography of the nodes, number of MPI's process, etc.) and is responsible for the parallelization of the simulation (managing the VRNGs and collecting the result on each MPI's process to allow the final computations).

Some classical designs introduce an object consisting of a Particles Factory which contains all the requirements for the particle simulations (the motion scheme or the diffusion and advection coefficients). The Launcher's role is then to distribute to each MPI's process a factory with the number of particles that must be simulated and the necessary VRNGs. The main job of the factory is to create objects which are considered as the particles and to store them. Each one of these objects contains all the necessary information for path simulation including the current time-dependent position and also the motion simulation algorithm.

This design is very interesting for interacting particles as it requires the storage of the path of each particle. For the case we decide to deal with, this implementation suffers two major flaws: a slowdown since many objects are created and a massive memory consumption as a large number of objects stay instantiated.

As a result, we decide to avoid the above approach and to use a design based on recycling. In fact, we choose to code a unique object that is similar to the factory, but does not create redundant particle objects. Let us call this object the Particle.

In few words, the recycling concept is the following. When the final position at time T is reached for each path, the Particle resets to the initial position and performs another simulation. This solution avoids high memory consumption and allows complete management of the memory. In addition, we do not use a garbage collector which can cause memory leaks.

Another thing, we adopt in our design, is the latest standards in the C++11 library [1] which offers the possibility to program an object with a template whose parameter is the spatial dimension of the process we want to simulate. Thus, one can include this template parameter into the implementation of the function governing the motion of the particle. If it is, the object is declared with the correct dimension and automatically changes the function template. Otherwise, it checks the compatibility of the declared dimension with the function.

Such a feature allows the ability to preallocate the exact size required by the chosen dimension for the position in a static array. Subsequently, we avoid writing multiple objects or using a pointer

and dynamic memory allocation, which provoke slowdown. Moreover, templates allow for a better optimization during the compilation.

Now a natural parallel scheme for a Monte Carlo simulation consists in the distribution of a particle on the different MPI's processes. Then, a small number of paths are sequentially simulated on each MP. When each MPI's process has finished, the data is regrouped on the master MPI process using MPI communications between the MPI's processes. Thus, the quantities of interest can be computed by the master MPI's process.

This scheme is typically embarrassingly parallel and can be used with both shared or distributed memory paradigm. Here we choose the distributed memory paradigm as it offers the possibility to use supercomputers based on SGI Altix or IBM Blue Gene technologies. Furthermore, if the path of the particles needs be recorded, the shared memory paradigm can not be used due to a very high memory consumption.

2.2. Random Number Generators

The main difficulty with the parallelization of the Monte Carlo method is to ensure the independence of all the random numbers split on the different MPI's processes. To be precise, if the same random numbers are used on two different processes, the simulation will end up with non-independent paths and the targeted quantities will be erroneous.

Various recognized RNGs such as RNGStreams [11], SPRNG [12] or MT19937 [13] offer the possibility to use VNRGs an can be used on parallel architectures. Recently, algorithms have been proposed to produce advanced and customized VRNGs with MRG32k3a and MT19937 [3].

In PALMTREE, we choose RNGStreams which possesses the following two imbricated subdivisions of the backbone generator MRG32k3a:

1. Stream: 2^{127} consecutive random numbers
2. Substream: 2^{76} consecutive random numbers

and the VRNGs are just the same MRG32k3a in different states (See Figure 1). Moreover, this RNG has already implemented VRNGs [11] and passes several statistical tests which can be found in TestU01 that ensure the independence of random numbers [9].

Now a possible strategy with RNGStreams is to use a stream for each new simulation of a moment as we must have a new set of independent paths and to use the 2^{51} substreams contain in each stream to allocate VRNGs on the FPU or to the objects for each moment simulation. This decision clearly avoids the need to store the state of the generator after the computations.

2.3. Strategy of Attachment to the FPUs (SAF)

An implementation of SAF with RNGStreams and the C++ design proposed in Subsection 2.1 is very easy to perform as the only task is to attach a VRNG to each MPI's process in the Launcher. Then the particles distributed on each MPI's process are simulated, drawing the random number from the attached VRNG.

Sometimes a selective replay may be necessary to capture some singular paths in order to enable a physical understanding or for debugging purposes. However, recording the path of every particle is a memory intensive task as keeping the track of the random numbers used by each particle. This constitutes a major drawback for this strategy. SAO is preferred in that case.

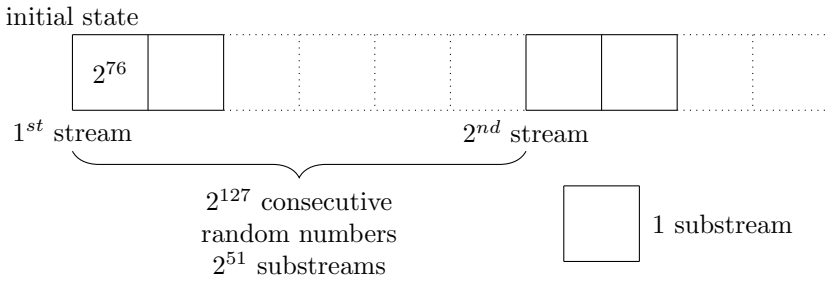


FIG 1. *The structure of RNGStreams*

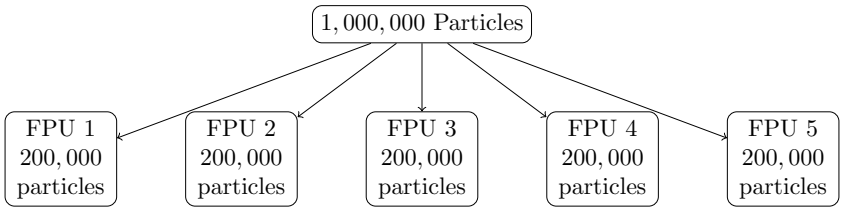


FIG 2. *Distribution of* $200,000$ *particles to each FPU*

2.4. Strategy of Object-Attachment (SAO) and PALMTREE

Here a substream is attached to each particle which can be considered as an object and all that is needed to implement this scheme is a subroutine to quickly jump from the first substream to the nth one. We show why in the following example: suppose that we need 1,000,000 paths to compute the moment and have 5 MPI's processes, then we distribute 200,000 paths to each MPI's process, which therefore requires 200,000 VRNGs to perform the simulations (See Figure 2).

The easiest way to solve this problem is to have the mth FPU that starts at the $(m-1) \times 200,000 + 1$st substream and then to jump to the next substream until it reaches the $m \times 200,000$th substream.

RNGStreams possesses a function that allows to go from one substream to the next one (See Figure 3). Thus the only problem is to go quickly from the first substream to the $(m-1) \times 200.000 + 1$st substream so that we can compete with the speed of the SAF.

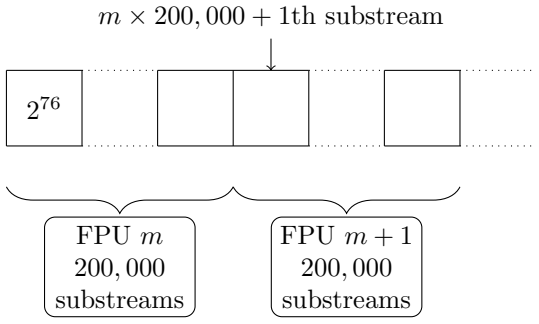


FIG 3. *Distribution of VRNGs or substreams to each FPU*

A naive algorithm using a loop containing the default function that passes through each substream one at a time is clearly too slow. As a result, we decide to modify the algorithm for MRG32k3a proposed in [3].

The current state of the generator RNGStreams is a sequence of six numbers, suppose that $\{s_1, s_2, s_3, s_4, s_5, s_6\}$ is the start of a substream. With the vectors $Y_1 = \{s_1, s_2, s_3\}$ and $Y_2 = \{s_4, s_5, s_6\}$, the matrix

$$A_1 = \begin{pmatrix} 82758667 & 1871391091 & 4127413238 \\ 36728315231 & 69195019 & 1871391091 \\ 3672091415 & 3528743235 & 69195019 \end{pmatrix}$$

and

$$A_2 = \begin{pmatrix} 1511326704 & 3759209742 & 1610795712 \\ 4292754251 & 1511326704 & 3889917532 \\ 3859662829 & 4292754251 & 3708466080 \end{pmatrix},$$

and the numbers $m_1 = 4294967087$ and $m_2 = 4294944443$, the jump from one substream to the next is performed with the computations

$$X_1 = A_1 \times Y_1 \mod m_1 \quad \text{and} \quad X_2 = A_2 \times Y_2 \mod m_2$$

with X_1 and X_2 the states providing the first number of the next substream.

As we said above, it is too slow to run these computations n times to make a jump from the 1st-substream to the nth-substream. Subsequently, we propose to use the algorithm developed in [3] based on the storage in memory of already computed matrix and the decomposition

$$s = \sum_{j=0}^{k} g_j \, 8^j,$$

for any $s \in \mathbb{N}$.

Since a stream contains $2^{51} = 8^{17}$ substreams, we decide to only store the already computed matrices

$$\begin{matrix} A_i & A_i^2 & \cdots & A_i^7 \\ A_i^8 & A_i^{2*8} & \cdots & A_i^{7*8} \\ \vdots & \vdots & \ddots & \vdots \\ A_i^{8^{16}} & A_i^{2*8^{16}} & \cdots & A_i^{7*8^{16}} \end{matrix}$$

for $i = 1, 2$ with A_1 and A_2 as above. Thus we can reach any substream s with the formula

$$A_i^s Y_i = \prod_{j=0}^{k} A_i^{g_j \, 8^j} \, Y_i \mod m_i$$

This solution provides a process that can be completed with a complexity less than $O(\log_2 p)$ which is much faster [3] than the naive solution. The Figure 4 illustrates this idea. In effect, we clearly see that the second FPU receive a stream and then performs a jump from the initial position of this stream to the first random number of the $n + 1$ substream of this exact same stream.

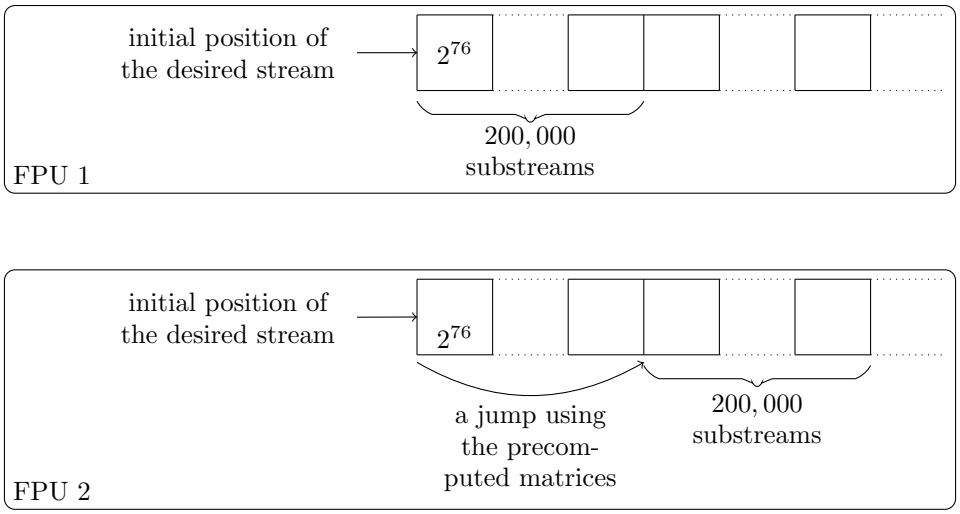


FIG 4. *Illustration of the stream repartition on FPUs*

3. Experiments with the Advection-Diffusion Equation

3.1. The Advection-Diffusion Equation

In physics, the solution $c(x,t)$ of (1) is interpreted as the evolution at the position x of the initial concentration $c_0(x)$ during the time interval $[0,T]$. The first moment of c is often called the center of mass.

Let us first recall that it exists a unique regular solution of (1). Proofs can be found [5, 14]. This clearly means, as we said in the introduction, that we deal with a well-posed problem.

The notion of fundamental solution [2, 4, 5, 14] which is motivated by the fact that $c(x,t)$ depends on the initial condition plays an important role in the treatment of the advection-diffusion equation. It is the unique solution $\Gamma(x,t,y)$ of

$$\begin{cases} \dfrac{\partial}{\partial t}\Gamma(x,t,y) = \mathrm{div}_x(\sigma(x)\cdot\nabla_x\Gamma(x,t,y)) - v(x)\nabla_x\Gamma(x,t,y), \\ \forall (x,t,y)\in \overline{D}\times[0,T]\times\overline{D}, \\ \Gamma(x,0,y) = \delta_y(x), \quad \forall (x,y)\in\overline{D}\times\overline{D}, \\ \Gamma(x,t,y) = 0, \quad \forall t\in[0,T], \quad \forall y\in\overline{D}, \quad \forall x\in\partial D. \end{cases} \tag{3}$$

This parabolic partial differential equation derived from (1) is often called the Kolmogorov Forward equation or the Fokker-Planck equation. The probability theory provides us with the existence of a unique Feller process $X = (X_t)_{t\geq 0}$ whose transition function density is the solution of the adjoint of (3), that is

$$\begin{cases} \dfrac{\partial}{\partial t}\Gamma(x,t,y) = \mathrm{div}_y(\sigma(y)\cdot\nabla_x\Gamma(x,t,y)) + v(y)\nabla_y\Gamma(x,t,y), \\ \forall (x,t,y)\in \overline{D}\times[0,T]\times\overline{D}, \\ \Gamma(x,0,y) = \delta_x(y), \quad \forall (x,y)\in\overline{D}\times\overline{D}, \\ \Gamma(x,t,y) = 0, \quad \forall t\in[0,T], \quad \forall x\in\overline{D}, \quad \forall y\in\partial D, \end{cases} \tag{4}$$

which is easy to compute since $\mathrm{div}(v(x)) = 0$ for every $x \in \mathbb{R}$.

Assuming that σ and v satisfy the hypotheses settled in (1), then using the Feynman-Kac formula [15] and (4), we can define the process X as the unique strong solution of the Stochastic Differential Equation

$$dX_t = v(X_t)\,dt + \sigma(X_t)\,dB_t, \tag{5}$$

starting at the position y and killed on the boundary D. Here, $(B_t)_{t\geq 0}$ is a d-dimensional Brownian motion with respect to the filtration $(\mathcal{F}_t)_{t\geq 0}$ satisfying the usual conditions [18].

The path of such a process can be simulated step-by-step with a classical Euler scheme. Therefore a Monte Carlo algorithm for the simulation of the center of mass simply consists in the computation until time T of a large number of paths and the average of all the final positions of every simulated particle still inside the domain.

As we are mainly interested in computational time and efficiency, the numerical experiments that will follow are performed in free space. Working on a bounded domain would only require to set the accurate stopping condition, which is a direct consequence of the Feynman-Kac formula that is to terminate the simulation of the particle when it leaves the domain.

3.2. Brownian Motion Simulation

Let us take an example in dimension one. We suppose that the drift term v is zero and that $\sigma(x)$ is constant. We then obtain the renormalized Heat Equation whose solution is the standard Brownian Motion.

Let us divide the time interval $[0, T]$ into N subintervals by setting $\delta t = T/N$ and $t_n = n \cdot \delta t$, $n = 0, ..., N$ and use the Euler scheme

$$X_{t_{n+1}} = X_{t_n} + \sigma\,\Delta B_n, \tag{6}$$

with $\Delta B_n = B_{t_{n+1}} - B_{t_n}$. In this case, the Euler scheme presents the advantage of being exact.

Since the Brownian motion is easy to simulate, we choose to sample 10,000,000 paths starting from the position 0 until time $T = 1$ with 0.001 as time step. We compute the speedup S and the efficiency E which are defined as

$$S = \frac{T_1}{T_p} \text{ and } E = \frac{T_1}{p\,T_p} \times 100,$$

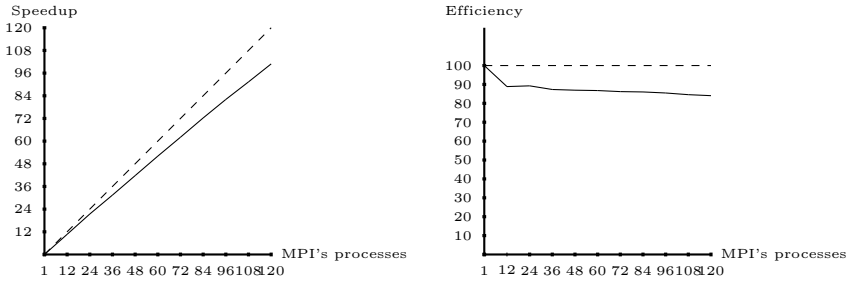
where T_1 is the sequential computational time with one MPI's process and T_p is the time in parallel using p MPI's process.

The speedup and efficiency curves together with the values used to plotted them are respectively given in Figure 5 and Table 1. The computations were realized with the supercomputer Lambda from the Igrida Grid of INRIA Research Center Rennes Bretagne Atlantique. This supercomputer is composed of 11 nodes with 2×6 Intel Xeon(R) E5647 CPUs at 2.40 Ghz on Westmere-EP architecture. Each node possesses 48 GB of Random Access Memory and is connected to the others through infiniband. We choose GCC 4.7.2 as C++ compiler and use the MPI library OpenMPI 1.6 as we prefer to use opensource and portable software. These tests include the time used to write the output file for the speedup computation so that we also show the power of the HDF5 library.

Processes	1	12	24	36	48	60	72	84	96	108	120
Time (sec.)	4842	454	226	154	116	93	78	67	59	53	48
Speedup	1	10.66	21.42	31.44	41.74	52.06	62.07	72.26	82.06	91.35	100.87
Efficiency	100	88.87	89.26	87.33	86.96	86.77	86.21	86.03	85.48	84.59	84.06

TABLE 1

The values used to plot the curve in Figure 5

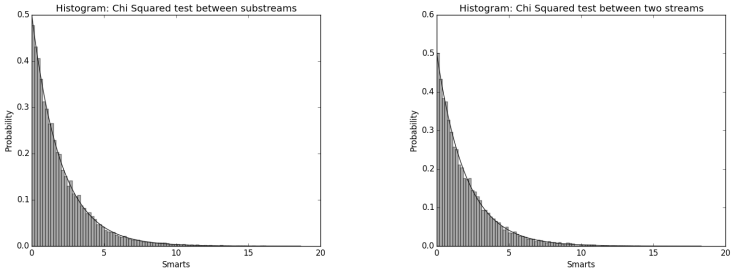


(a) Speedup (b) Efficiency

FIG 5. *Brownian motion: (a) The dash line represents the linear acceleration and the black curve shows the speedup. (b) The dash line represents the 100% efficiency and the black curve shows the PALMTREE's efficiency.*

The Table 1 clearly illustrates PALMTREE's performance. It appears that the SAO does not suffer a significant loss of efficiency despite it requires a complex preprocessing. Moreover, the data show that the optimum efficiency (89.26%) is obtained with 24 MPI's processes.

As we mentioned in Subsection 2.2, the independence between the particles is guaranteed by the non correlation of random numbers generated by the RNG. Moreover, Figure 6 shows that the sum of the squares of the positions of the particles at $T = 1$ follow a χ^2 distribution in two different cases: (a) between substreams i and $i + 1$ for $i = 0, \ldots, 40000$ of the first stream. (b) between substreams i of the first and second streams for $i = 0, \ldots, 10000$.

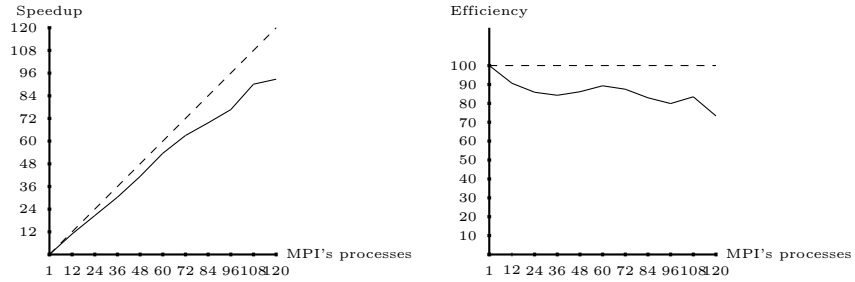


(a) Substreams (b) Streams

FIG 6. χ^2 *test: (a) between substreams* i *and* $i + 1$ *for* $i = 0 \ldots 40000$ *of the first stream. (b) between substreams* i *of the first and second streams for* $i = 0 \ldots 10000$.

Processes	1	12	24	36	48	60	72	84	96	108	120
Time (sec.)	19020	1749	923	627	460	355	302	273	248	211	205
Speedup	1	10.87	20.60	30.33	41.34	53.57	62.98	69.67	76.69	90.14	92.78
Efficiency	100	90.62	85.86	84.26	86.14	89.29	87.47	82.94	79.88	83.46	73.31

TABLE 2
The values used to plot the curve in Figure 7



(a) Speedup (b) Efficiency

FIG 7. *Constant diffusion with an affine drift: (a) The dash line represents the linear acceleration and the black curve shows the speedup. (b) The dash line represents the 100% efficiency and the black curve shows the PALMTREE's efficiency.*

3.3. Advection-Diffusion Equation with an Affine Drift Term

We now consider the advection-diffusion equation whose drift term v is an affine function, that is for each $x \in \mathbb{R}$, $v(x) = a\,x + b$ and σ is a constant. We simulate the associated stochastic process X through the exact scheme

$$X_{t_{n+1}} = e^{a\delta t} X_{t_n} + \frac{b}{a}(e^{a\delta t} - 1) + \sigma\sqrt{\frac{e^{2a\delta t} - 1}{2a}}\mathcal{N}(0,1)$$

where $\mathcal{N}(0,1)$ is a standard Gaussian law [8].

For this scheme with an initial position at 0 and the parameters $\sigma = 1$, $a = 1$, $b = 2$ and $T = 1$, we give the speedup and efficiency curves represented in Figure 7 based on the simulation of hundred millions of particles. The Table 2 provides the data resulting from the simulation and used for the plots.

Whatever the number of MPI's processes involved, we obtain the same empirical expectation $\mathbb{E} = 3.19$ and empirical variance $\mathbb{V} = 13.39$ with a standard error $S.E. = 0.0011$ and a confidence interval $C.I. = 0.0034$. Moreover, a good efficiency (89.29%) is obtained with 60 MPI's processes.

In this case, the drift term naturally pushes the particles out of 0 relatively quickly. If this behavior is not clearly observed in a simulation, then the code has a bug and a replay of a selection of few paths can be useful to track it in spite of reviewing all the code. This can clearly save time.

With the SAO, this replay can be easily performed since we know which substreams is used by each particle as it is shown in Figure 4. Precisely, in the case presented in Figure 2 and 3, the nth particle is simulated by a certain FPU using the nth substream. As a result, it is easy to replay the nth particle since we just have to use the random numbers of the nth substream. The point is that the parameters must stay exactly the same particularly the time step. Otherwise, the replay of

the simulation will use the same random numbers but not for the exact same call of the generator during the simulation.

4. Conclusion

The parallelization of Stochastic Lagrangian solvers relies on a careful and efficient management of the random numbers. In this paper, we proposed a strategy based on the attachment of the Virtual Random Number Generators to the Object.

The main advantage of our strategy is the possibility to easily replay some particle paths. This strategy is implemented in the PALMTREE software. PALMTREE use RNGStreams to benefit from the native split of the random numbers in streams and substreams.

We have shown the efficiency of PALMTREE on two examples in dimension one: the simulation of the Brownian motion in the whole space and the simulation of an advection-diffusion problem with an affine drift term. Independence of the paths was also checked.

Our current work is to perform more tests with various parameters and to link PALMTREE to the platform H2OLAB [16], dedicated to simulations in hydrogeology. In H2OLAB, the drift term is computed in parallel so that the drift data are split over MPI's processes. The challenge is that the computation of the paths will move from one MPI's process to another which raises issues about communications, good work load balance and an advanced managemement of the VRNGs in PALMTREE.

References

[1] The c++ programming language. *https://isocpp.org/std/status*, 2014.

[2] D. G. Aronson. Non-negative solutions of linear parabolic equations. *Annali della Scuola Normale Superiore di Pisa - Classe di Scienze*, 22(4), 607–694, 1968.

[3] T. Bradley, J. du Toit, M. Giles, R. Tong, P. Woodhams. Parallelization Techniques for Random Number Generations. *GPU Computing Gems Emerald Edition*, 16, 231–246, 2011.

[4] L. C. Evans. *Partial Differential Equations*. Graduate studies in mathematics, vol. 19, second edition, American mathematical society, 2010.

[5] A. Friedman. *Partial Differential Equations of Parabolic Type*. Dover Books on Mathematics Series, Dover Publications, 2008.

[6] C. Gardiner. *A Handbook for the Natural and Social Sciences*. Springer Series in Synergetics, vol. 13, fourth edition, Springer-Verlag Berlin Heidelberg, 2009.

[7] W. Hundsdorfer, J. G. Verwer. *Numerical Solution of Time-Dependent Advection-Diffusion-Reaction Equations*. Springer Series in Computational Mathematics, vol. 33, Springer-Verlag Berlin Heidelberg, 2003.

[8] P. E. Kloeden, E. Platen. *Numerical Solution of Stochastic Differential Equations*. Stochastic Modelling and Applied Probability, vol. 23, Springer-Verlag Berlin Heidelberg, 1992.

[9] P. L'Ecuyer. Testu01. *http://simul.iro.umontreal.ca/testu01/tu01.html*.

[10] P. L'Ecuyer, D. Munger, B. Oreshkin, R. Simard. Random Numbers for Parallel Computers: Requirements and Methods, With Emphasis on GPUs. *Mathematics and Computers in Simulation*, revision submitted, 2015.

[11] L'Ecuyer, P., Simard, R., E. J. Chen, W. D. Kelton. An Object-Oriented Random-Number Package with Many Long Streams and Substreams. *Opererations Research*, 50(6), 1073–1075, 2002.

[12] M. Mascagni, A. Srinivasan. Algorithm 806: Sprng: A Scalable Library for Pseudorandom Number Generation. *ACM Transactions on Mathematical Software* 26(3), 436–461, 2000.

[13] M. Matsumoto, T. Nishimura. Mersenne twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1), 3–30, 1998.

[14] J. Nash. Continuity of Solutions of Parabolic and Elliptic Equations. *American Journal of Mathematics*, 80(4), pp. 931–954, 1958.

[15] B. Øksendal. *Stochastic Differential Equations*. Universitext, sixth edition, Springer-Verlag Berlin Heidelberg, 2003.

[16] Project-team Sage. H2OLAB. *https://www.irisa.fr/sage/research.html*.

[17] L. Lenôtre, G. Pichot. Palmtree library. *http://people.irisa.fr/Lionel.Lenotre/software.html*.

[18] D. Revuz, M. Yor. *Continuous Martingales and Brownian Motion*. Grundelehren der mathematischen Wissenschaften, vol. 293, third edition, Springer, Berlin, 1999.

[19] C. Zheng, G. D. Bennett. *Applied Contaminant Transport Modelling*. Wiley, 2002.