A Strategy for Parallel Implementations of Stochastic Lagrangian Simulation

Lionel Lenôtre

A Strategy for Parallel Implementations of Stochastic Lagrangian Simulation

Lionel Lenôtre

June 30, 2015

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 shows 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 becomes a very convenient method to solve problems arising in physics like the advection-diffusion equation with a Dirichlet boundary condition

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

where, for each $x \in \mathbb{R}$, $\sigma(x)$ is a d-dimensional square matrix, $v(x)$ is a d-dimensional vector such that $\operatorname{div}(v(x)) = 0$, $D \subset \mathbb{R}^d$ is a regular open bounded subset and T is a positive real number. In order to have a well-posed problem, we required that, for each $x \in \mathbb{R}$, $\sigma(x)$ is definite, positive, symmetric and satisfy a condition of ellipticity (See [4, 5] for example).

Interesting computations involving the solution $c(t,x)$ are the moments at a time T

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

One possibility for the computation of these moments 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 in order 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, 20] which is really simple. Indeed, the theory of stochastic process implies that it exists a stochastic process $X = (X_t)_{t \geq 0}$ 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 to 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 adoption of supercomputers and parallel architectures becomes a key ingredient to obtain reasonable computational times. However, the main difficulty when one implements for 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 mean by Virtual Random Number Generators (VRNGs) either different independent Random Number Generators (RNGs) or copies of the same RNG in different states [9].

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 VRNG. Note that 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 VRNG.

Both schemes clearly ensure the non correlation of the particles assuming that all the drawn random numbers has enough independence which is clearly a matter of RNG.

Sometimes particles with a singular behavior are encountered and the examination of the full paths of such particles is necessary. With 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 SAO. We developed a library, called PALMTREE [16], that implements the Monte-Carlo method with a management of the random numbers according to SAO.

The present paper is organized in two sections. The first one describes SAF and SAO. Broadly speaking, we will treat of the distribution of the VRNGs and the work done in PALMTREE with the generator RNGStreams [10]. The second section presents two numerical experiments which illustrate the performance of PALMTREE. They both provide characteristic curves like speedup and efficiency.

2 Parallel and Object-Oriented implementations in Monte Carlo

All along this section, we will assume that we are able to simulate the transition law of particles undergoing a Markovian dynamic such that there is no interaction between them. As a result, the presentation below can be applied to various Monte Carlo scheme involving particles tracking where the goal is to compute moments. Moreover, this shows the flexibility of the implementation we chose.

2.1 An Object-Oriented design for Monte Carlo

C++ offers very interesting features which is of great help for a fast execution and d-dimensional process. 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 will describe and motivate the choices we made in the implementation of PALMTREE. Note that we wil 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 like 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, like 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 responsibility 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 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 since a large number of objects stay instantiated.

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. All along the paper, we will refer to this object as the Particle. 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 provoke memory leaks.

The latest standards in the C++11 library [3] 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. As a result, we avoid writing multiple objects or using a pointer and dynamic memory allocation, which provoke slowdown. Furthermore, templates allow for a better optimization during the compilation.

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 communication 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 must be recorded, the shared memory paradigm can not be used due to a very high memory consumption.

2.2 Random Number Generators

Various recognized RNGs such as RNGStreams [10], SPRNG [12] or MT19937 [13] offer the possibility to use VNRGs. Recently, algorithms have been proposed to produce advanced and customized VRNGs with MRG32k3a and MT19937 [2].

RNGStreams 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.

A strategy with RNGStreams is to use a Stream for each new simulation of an expectation, as we must have a new set of independent paths. This decision avoids the need to store the state of the generator after the computations, since we use a VRNG for each new simulation. Inside each Stream, we can use 2^{51} substreams.

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.

In PALMTREE, we choose RNGStreams as this RNG has already implemented VRNGs [10] and passes several statistical tests which can be found in TestU01 that ensure the independence of random numbers [11] .

2.3 Strategy of attachment to the FPUs (SAF)

An implementation of SAF with RNGStreams and the C++ design proposed in 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 Particle on each MPI's process runs the simulation, 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.

2.4 Strategy of Object-Attachment (SAO) and PALMTREE

Here the substream is attached to the Particle. Then, we can save computational time by replaying them quickly in a new simulation.

All that is needed to implement this scheme is a subroutine to quickly jump from the first substream to the pth one. We show why in the following example: suppose that we need 1,000,000 paths to compute the moment with 10 MPI's processes, then we need to give 100,000 paths to each MPI's process, which requires 100,000 of VRNGs to perform the simulations.

The easiest way to implement this example is to have the nth process that starts at the $n \times 100.000 + 1$th substream, and to jump to the next substream until it reaches the $n+1 \times 100.000$th substream (since RNGStreams possesses a function that allows to go from one substream to the next). The only problem is to go quickly from the first substream to the $n \times 100.000 + 1$th substream so that we can compete with the computation time of the SAF. 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 choose to modify the algorithm for MRG32k3a proposed in [2]. 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

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

and

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

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 state providing the first number of the next substream. As we said above, it is too slow to run these computations p times to reach the pth-substream. Thus, we propose to use the algorithm developed in [2] based on the storage in memory of already computed matrix and the decomposition

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

for any $p \in \mathbb{N}$. Since a Stream contains $2^{51} = 8^{17}$ substreams, we decide to only store the matrix

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

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

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

This solution provides a process that can be completed with a complexity less than $O(\log_2 p)$ which much faster than a naive solution.

3 Numerical Experiments with advection-diffusion equations

3.1 A reminder on 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 $[0,T]$ and the first moment is often called the center of mass.

The first fact we want to recall is the existence and uniqueness of a regular solution of (1) whose proofs can be found [5, 14]. This means, as said in the introduction, that the problem is well-posed.

The second thing is the notion of fundamental solution which is motivated by the fact that $c(x,t)$ depends on the initial condition. More precisely, the fundamental solution $\Gamma(x,t,y)$ is the unique solution of

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

See [1, 4, 5, 14] for more details and results on existence and uniqueness of $\Gamma(x,t,y)$.

The above parabolic partial differential equation derived from (1) is called the Kolmogorov Forward equation or Fokker-Planck equation. As we mentioned in the introduction, the theory of probability provides the existence of a unique Feller process $X = (X_t)_{t\geq 0}$ such that the density of the transition function starting from y is the solution of the adjoint equation

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

which, in the context of the theory of semigroup, is equivalent to the infinitesimal generator of the process X [15, 18, 19]. We will refer to (4) as the Kolmogorov Backward Equation. Note that the adjoint is easy to compute since $\mathrm{div}(v(x)) = 0$ for every $x\in\mathbb{R}$.

If we suppose that σ and v are continuous and admit a derivative on $\overline{D}$, then using the Feynman-Kac formula [15] and (4) we are able to 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 $(\mathscr{F}_t)_{t\geq 0}$ satisfying the usual conditions [18]. Note that the hypothesis on σ and v settled in (1) ensure the existence of a unique strong solution to (6). See [8, 18] for further details.

The path of such a process can be simulated step-by-step with a classical Euler scheme.

An algorithm of the Monte Carlo simulation for the center of mass consists in the computation until the time T of a large number of paths and taking the average of all the final positions of every simulated particle which are 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 left the domain.

3.2 Brownian motion simulation

Let us take a one-dimensional example. Suppose the drift term is zero and $\sigma(x)$ is constant in problem (4). Then we obtain the renormalized Heat Equation whose solution is the standard Brownian Motion.

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

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

with $\Delta B_n = B_{t_{n+1}} - B_{t_n}$, which presents the advantage of being exact.

Since the Brownian motion is easy to simulate, we choose to sample a large number of paths (10,000,000) starting from the position 0 with a small timestep (0.001) until time $T = 1$. We compute the speedup S and the efficiency E defined by

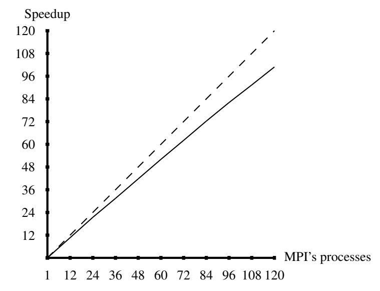
$$S = \frac{T_1}{T_p} \text{ and } E = \frac{T_1}{p\,T_p}\times 100,$$

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.

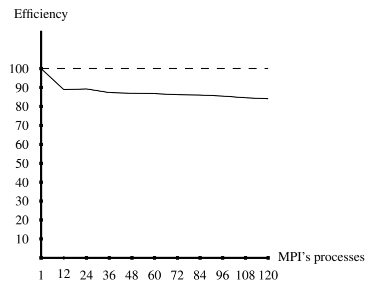
The speedup curve represented in fig.1 was realized with the supercomputer Lambda from the Igrida Grid of INRIA Research Center Rennes Bretagne Atlantique. This supercomputer has 11 nodes with 2×6 Intel Xeon(R) E5647 CPUs at 2.40 Ghz on Westmere-EP architecture. Each node is equipped with 48 GB of RAM 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., since 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

This array 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.



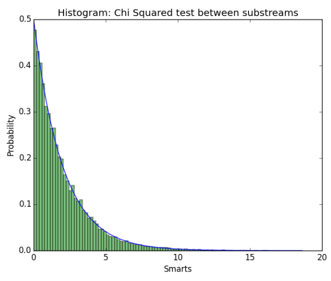
(a) Speedup



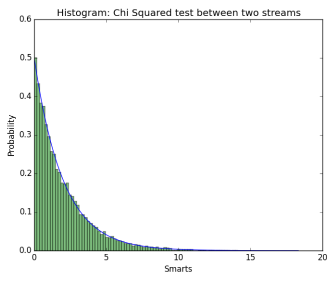
(b) Efficiency

Figure 1: 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.

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, fig. 2 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 $i = 0\dots40000$ of the first stream. (b) between substreams i of the first and second streams for $i = 0\dots10000$.



(a) Substreams



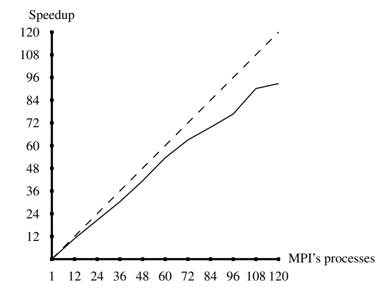
(b) Streams

Figure 2: χ^2 test: (a) between substreams i and $i+1$ for $i = 0\dots40000$ of the first stream. (b) between substreams i of the first and second streams for $i = 0\dots10000$.

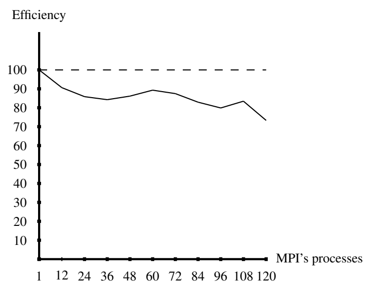
3.3 Advection-diffusion equation with an affine drift term

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

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



(a) Speedup



(b) Efficiency

Figure 3: 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.

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 fig. 3 based on the simulation of hundred millions of particles. The array below 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 interval of confidence $I.C. = 0.0034$. Moreover, a good efficiency (89.29%) is obtained with 60 MPI's processes.

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

4 Conclusion

The parallelization of Stochastic Lagrangian solvers relies on a careful and efficient management of the random numbers. We proposed a strategy based on the attachment of the Virtual Random Numbers to the Object and an algorithm that allows a quick jump within the random numbers. One advantage is an easy replay of the particle paths. This strategy is implemented in the PALMTREE software. PALMTREE use RNGStreams as RNG to benefit from the split of the random numbers in Streams and Substreams.

We have shown the efficiency of PALMTREE on two one-dimensional examples: 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 [17], 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] 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.

[2] T. Bradley, J. du Toit, R. Tong, M. Giles and P. Woodhams. Parallelization techniques for random number generations. *GPU Computing Gems Emerald Edition, Morgan Kaufmann*, 16: 231–246, 2011.

[3] The C++ Programming Language. *https://isocpp.org/std/status*, 2014.

[4] L.C. Evans. Partial differential equations. Graduate studies in mathematics, volume 19, second edition, American mathematical society, 2010.

[5] A. Freedman. Partial Differential Equations of Parabolic Type. *Englewood Cliffs, N.j.: Prentice-Hall*, 1964.

[6] C. Gardiner. Stochastic Methods: A Handbook for the Natural and Social Sciences. Springer Series in Synergetics, Springer Complexity, 2009.

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

[8] P. E. Kloeden and E. Platen. Numerical solution of stochastic differential equations. Applications of Mathematics, Stochastic Modelling and Applied Probability, volume 23, 1999.

[9] P. L'Ecuyer, B. Oreshkin and R. Simard. Random Numbers for Parallel Computers: Requirements and Methods. to appear.

[10] P. L'Ecuyer, R. Simard, E. J. Chen and W. D. Kelton. An oriented object random-number generator package with many long streams and substreams. 2000.

[11] P. L'Ecuyer. TestU01. http://simul.iro.umontreal.ca/testu01/tu01.html.

[12] M. Mascagni and A. Srinivasan. Algorithm 806: SPRNG: A scalable library for pseudorandom number generation. *ACM Transactions on Mathematical Software*, 26:436–461, 2000.

[13] M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Mathematical Software*, 8: 3–30, 1998.

[14] J. F. Nash. Continuity of Solutions of Parabolic and Elliptic Equations. *American Journal of Mathematics*, 80(4):931–964, 1958.

[15] B. Oksendal. Stochastic Differential Equations, sixth edition. *Universitext, Springer*, 2007.

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

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

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

[19] D. W. Stroock. Diffusion semigroups corresponding to uniformly elliptic divergence form operator. *Séminaire de probabilités (Strasbourg)*, 22: 316–347,n 1988.

[20] C. Zheng and G. D. Bennett Applied Contaminant Transport Modelings, second edition. *Wiley-Interscience*, 2002.