A Strategy for the Parallel Implementations of Stochastic Lagrangian Methods

Lionel Lenôtre

A Strategy for Parallel Implementations of Stochastic Lagrangian Simulation

Lionel Lenôtre

September 19, 2014

Abstract

In this paper, we present some investigations on the parallelization of a stochastic Lagrangian simulation. For the self sufficiency of this work, we start by recalling the stochastic methods used to solve Parabolic Partial Differential Equations with a few physical remarks. Then, we exhibit different object-oriented ideas for such methods. In order to clearly illustrate these ideas, we give an overview of the library PALMTREE that we developed. After these considerations, we discuss the importance of the management of random numbers and argue for the choice of a particular strategy. To support our point, we show some numerical experiments of this approach, and display a speedup curve of PALMTREE. Then, we discuss the problem in managing the parallelization scheme. Finally, we analyze the parallelization of hybrid simulation for a system of Partial Differential Equations. We use some works done in hydrogeology to demonstrate the power of such a concept to avoid numerical diffusion in the solution of Fokker-Planck Equations and investigate the problem of parallelizing scheme under the constraint entailed by domain decomposition. We conclude with a presentation of the latest design that was created for PALMTREE and give a sketch of the possible work to get a powerful parallelized scheme.

1 Introduction

Monte Carlo Simulation (MCS) becomes a very convenient method to solve Parabolic Partial Differential Equation (PPDE). A well known example of PPDE, which will be used all along this paper, is the Convection-Diffusion Equation with Dirichlet boundary condition:

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

Here, σ is a d-dimensional square matrix, v is a d-dimensional vector, $D \subset \mathbb{R}^d$ is a regular open bounded subset and T is a positive real number. This equation is used in physics to describe density dynamics in a material undergoing diffusion and

convection. More than the solution, physicists are interested by some quantities of interest like the center of mass:

$$c_m = \int_D c(t,x)\,dx.$$

The quantity c_m can be computed by numerical integration of an approximated solution of Equation 1, which comes from a solver like Finite Difference Method, Finite Volume Method or Finite Element Method. Such a procedure belongs to the family of Eulerian Methods (EM). The problem with EM is that they introduce numerical diffusion [9]. In practice, physicists use MCS in order to eliminate such numerical diffusion errors [8, 19]. The computation of c_m using MCS consists of the average value of the positions at time T of a very large number of particles whose motion is described by the adjoint of the equation. In fact, the solution of the adjoint is the density of the distribution governing the motion of the particles. The main problem with MCS is the slow rate of convergence caused by the Central-Limit Theorem. For example, the computation of the center of mass requires a large amount of particles to achieve a reliable approximation. Thus, the adoption of supercomputers and parallel architectures becomes a key to obtain reasonable computational times.

In this paper, we investigate the parallelisation of the MCS for the center of mass. The main difficulty is to manage the random numbers such that the particles are not correlated, thus avoiding a bias in the approximation. A classical implementation strategy relies on the distribution to each Arithmetic and Logic Unit (ALU) of Virtual Random Number Generators (VRNGs) which are different independent Random Number Generators (RNGs) or copies of the same RNG in different states [10]. Then, the total amount of particles is divided into small batches which are run on the different ALUs. Such a scheme clearly ensures the non correlation of every particles, as all the drawn random numbers are independent. We refer to this scheme as the Strategy of Attachment to the ALUs (SAA). A problem arises when particles with divergent behaviour are encountered and the examination of the full paths of such particles is necessary for debugging or physical understanding. In fact, recording the path of every particle is a memory intensive task and is avoided during simulation. Thus, a replay of a simulation is required and every particle should be relaunched with a condition implemented to record only the necessary particles. One way to avoid reproducing the simulation of every particle is to keep track of the random numbers used by each particle, which would drastically increase the computational time and add unnecessary complications to the code. In order to solve the problem with replay, we choose to present a scheme which consist of attributing a different VRNG to each particle and distributing the particles to the different ALUs. We call this scheme the strategy of Attachement to the Object (SAO) and propose an implementation in the library PALMTREE. We show tests of our implementation and detail the problems which arise with the hybrid simulation of a system of coupled equations where where both Deterministic Eulerian simulation and Lagrangian Stochastic Simulation are used to compute the center of mass. We will detail this particular problem in the last section.

This paper is arranged in three parts. For self sufficiency, the first one starts with a reminder of the MCS for PPDE. A practical methodology for the implementation in parallel of such a simulation follows. The presented methodology is the one adopted in PALMTREE. This second part furnish a full explanation of our parallelisation strategy, the distribution of the VRNGs and the work done in PALMTREE

with the generator RNGStreams [11]. The part end up with speedup plots in various context in order to show the relevance of our strategy. In the last part, we explain the notion of hybrid simulation for a system of coupled equation. We exhibit an example in hydrogeology of such a system and the implementation of a simulation which comes from the platform H2OLAB[2, 6]. This example allows us to show the flaws of our presented strategy and to start thinking of solutions.

2 Simulation of the Convection-Diffusion Equation

2.1 A Theoretical Reminder

In physics, the solution $c(x,t)$ of Equation 1 is interpreted as the evolution at the position x of the initial concentration $c_0(x)$ during the time interval $[0,T]$. Note that such an equation admit a unique regular solution [7, 15]. Moreover, this equation belongs to the family of Fokker-Planck Equations. As the classical solution of such equations depends on the initial condition, we introduce the notion of the fundamental solution $\Gamma(x,t,y)$ which is defined for all $(x,t) \in \overline{D} \times [0,T]$, $y \in \overline{D}$, and satisfies the following conditions:

1. as a function of (x,t), $x \in D$, $t \in]0,T]$, we have

$$\frac{\partial}{\partial t}\Gamma(x,t,y) = \sigma \Delta_x \Gamma(x,t,y);$$

2. for every continuous function $f(x) \in \overline{D}$, if $x \in D$ then

$$\lim_{t \to 0} \int_D \Gamma(x,t,y) f(y)\, dy = f(x).$$

The fundamental solution of equation 1 exists and is unique [1, 15]. Furthermore, the physical interpretation remains the same, as we just have replace initial condition c_0 by the Dirac mass at y and the fundamental solution can be define as the solution of

$$\begin{cases} \dfrac{\partial}{\partial t}\Gamma(x,t,y) = \mathrm{div}_x(\sigma \cdot \nabla_x \Gamma(x,t,y)) - \nabla_x(v\Gamma(x,t,y)), \\ \forall (x,t,y) \in \overline{D} \times [0,T] \times \overline{D}, \\ \Gamma(x,0,y) = \delta_y(x), \quad \forall (x,y) \in \overline{D} \times \overline{D}, \\ \Gamma(x,t,y) = 0, \quad \forall t \in [0,T], \quad \forall y \in \overline{D}, \quad \forall x \in \partial D, \end{cases} \tag{2}$$

Such a PPDE derived from equation 1 is called a Kolmogorov Forward Equation (KFE) and probability theory provides the existence of a unique Feller process $(X_t)_{t \geq 0}$ such that the density of the transition function is the solution of the adjoint of the above equation [17, 18]. We refer to this adjoint as the Kolmogorov Backward Equation (KBE). For the physicist, such an equation describes the evolution of the position of a particle which moves undergoing the diffusion tensor σ and the convection vector v. Note that we have switch from a Eulerian to a Lagrangian

description. The adjoint of Equation 2 is the following:

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

In the context of the theory of semigroup, the KBE is equivalent to the infinitesimal generator [16, 17, 18]. With KBE and the Feynman-Kac formula [16], we can define the process $(X_t)_{t\geq 0}$ as the solution $(Y_t)_{t\geq 0}$ of the Stochastic Differential Equation (SDE):

$$dY_t = v\,dt + \sigma\,dB_s, \tag{4}$$

starting at the position x and killed on the boundary D. Here, $(B_s)_{s\geq 0}$ is a d-dimensional Brownian motion with respect to the filtration $(\mathscr{F}_s)_{s\geq 0}$ satisfying the usual conditions [17]. The path of such a process can be simulated step-by-step with classical Euler scheme and the accurate stopping condition. As a result, an algorithm of the MCS for the center of mass consists of the computation until the time T of a large number of paths with Euler scheme and the average of all the final positions of every simulated particle which are still inside the domain.

2.2 An Object-Oriented Implementation Strategy

Object-Oriented Programming (OOP) languages like C++ are a key for MCS's code as they offer very interesting features for code design. In order to discuss these features for MCS, we propose to review the C++ implementation of PALMTREE. This presentation is dictated by the fact that a good C++ design is key for a suitable parallel simulation code with a low computational time. Furthermore, C++ offers a consistent implementation of MPI, since we decide to use the distributed memory paradigm in PALMTREE for reasons that will be explained later in 3.1. To fit with the standard vocabulary of MPI, we we will refer to an ALU as a MPI Process (MP). In PALMTREE, we choose to design an object called the Launcher which conducts the MCS. This object 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 Process, etc.), and is responsible for the parallelisation of the simulation (managing the VRNGs and collecting the result on each MP to allow the final computations). Some classical OOP designs introduce an object consisting of a Particles Factory (PF). Such an object contains all the requirements for the particle simulations, like the motion scheme or the diffusion and convection coefficients. The Launcher's role is then to distribute to each MP a PF with the number of particles that must be simulated and the necessary VRNGs. The main responsibility of the PFs 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 motion algorithm and the current time-dependent position. This design is very interesting for interacting particles, since it requires the storage of the path of each particle. For convection-diffusion equation, this implementation suffers two major flaws: a slowdown as many objects are created and massive memory consumption since a large number of objects stay instantiated. We decide to avoid this approach in PALMTREE, and to use a design based on recycling. In fact, we choose to code a

unique object that is similar to the PF, but does not create redundant particle objects. Instead, as the final position at time T is reached for each path, it 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. We will refer to this object as the Particle. Another reason for the adoption of C++ in spite of other OOP languages relies on the use of one of the latest standards in the C++11 library [5]. This standard offers the possibility to program an object with a template whose parameter is the spatial dimension of the convection-diffusion equation 2. 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 compatability 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.

3 Parallelisation Schemes for Monte Carlo Simulation

3.1 VRNG and Basic Parallelisation Scheme

A natural parallel scheme for a MCS consists of the distribution of a Particle on the different MPs. Then, a small number of paths are sequentially simulated on each MP. When each MP has finished, the data is regrouped on the master MP using MPI communication between the MPs. Thus, the quantities of interest can be computed by the master MP. 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 every particle must be recorded, the shared memory paradigm can not be used due to a very high memory consumption. With the above scheme and the chosen paradigm, the main difficulty is to ensure the independence of all the random numbers split on the different MPs. To be precise, if the same random numbers are used on two different MPs, the simulation will end up with non-independent paths and the targeted quantities will be erroneous. Thus, random number management becomes the main challenge and VRNGs are key. Various recognized RNGs such as RNGStreams [11], SPRNG [13] or MT19937 [14] offer the possibility to use VNRGs. Recently, many algorithms have been proposed to produce advanced and customized VRNGs with MRG32k3a and MT19937 [4]. In PALMTREE, we choose RNGStreams as this RNG has already implemented VRNGs [11]. More precisely, 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. An implementation of the SAA with RNGStreams and the C++ design proposed in 2.2 is very easy to

perform, as the only task is to attach a VRNG to each MP in the Launcher. Then, the Particle on each MP runs the simulation, drawing the random number from the attached VRNG. We have already described the problem that arises with such a scheme. Since recording the path of every particle is a memory intensive task and is avoided during simulation, a selective replay is necessary to capture some divergent paths in order to enable a physical understanding or for debugging purposes. As we said in the introduction, a solution is to keep track of the random numbers used by each path, but that would drastically increase the computational time and add unnecessary complications to the code.

3.2 PALMTREE and Object-Attachment Scheme

In PALMTREE, we decide to use a Stream for each new simulation of the center of mass, as we need a new set of independent paths. This decision avoids needing to store the state of the generator after the computation, since we use a VRNG for each new simulation. Inside each Stream, we can use 2^{51} substreams to parallelize the simulation of one center of mass. In this way, we choose a scheme totally different from the SAA, which consists of attributing a VRNG for each path simulation. But like in the SAA, we distribute a small number of paths in order to simulate on each MP. Since we store the substream of each path in our method, 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 center of mass with 10 MPs, then we need to give 100,000 paths to each MP, which requires 100,000 of VRNGs to perform the simulations. The easiest way to implement this example is to have the nth MP start 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 it 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 SAA. 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 [4]. The current state of the generator RNGStream 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 [4] 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 $2^{51} = 8^{17}$, 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 substreams with the formula

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

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.

3.3 Numerical Experiments

In order to show numerical experiments of the library PALMTREE, we choose to simulate the standard Brownian Motion (BM) in parallel. The BM is a the solution of the renormalized Heat Equation which is a Fokker-Planck Equation with a diffusion coefficient but without a drift term. In addition, we decide to solve the equation in free space (not inside a domain) and until the stopping time $T = 1$. Since the BM is easy to simulate, we choose to sample a large number of paths (10,000,000) with a small timestep (0.001). These values give us the possibility to exhibit a relevant speedup curve. For the self sufficiency of this paper, we give the formula of the speedup S:

$$S = \frac{T_1}{T_p}.$$

Here, T_1 is the sequential computational time with one MP and T_p is the time in parallel using p MPs. The speedup curve presented below (see 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 with 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 may also show the power of the HDF5 library. Our test results on a single node are:

Nb of cores	1	2	3	4	5	6	7	8	9	10	11	12
Time in sec.	4935	2330	1778	1316	1059	903	785	683	614	580	573	497

With the 10 nodes of 12 cores, we get:

Nb of nodes	1	2	3	4	5	6	7	8	9	10
Time in sec.	497	242	172	126	104	88	76	65	57	55

The above formula gives us the following speedup, where we only use a part of the above outcomes:

Nb of cores	2	4	6	8	10	12	24	36	48	96	120
Speedup	2.11	3.74	5.46	7.22	8.50	9.92	20.38	28.69	39.16	75.92	89.72

Note the super linear acceleration with 2 cores, which is due to a better memory management with 2 cores and caused by the Westmere-EP architecture. From the speedup, we derive the efficiency in percent:

Nb of cores	2	4	6	8	10	12	24	48	96	120
Efficiency	105	93	91	90	85	82	84	81	79	74

calculated with the formula:

$$E = \frac{T_1}{pT_p} \times 100.$$

This last array illustrates the SAO's performance, as it does not suffer a significant loss of efficiency. We recall that SAO requires more complex preprocessing, since we need to manage the distribution of the VRNGs. The more cores we have, the more complicated is the distribution, as we require more jumps from the first substream. This is the reason for the slowdown in the speedup figure. In conclusion, it appears to us that it does not seem relevant to launch a complicated benchmark test between the SAO and the SAA.

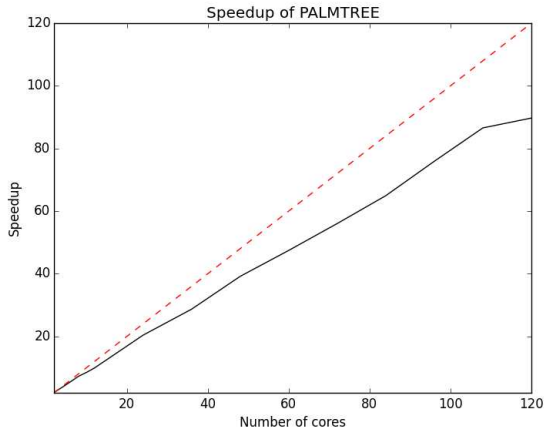

Figure 1: The dash red line represents the linear acceleration and the black curve shows the speedup.

4 Hybrid Simulation and Parallel Scheme

4.1 Hybrid Simulation for System of Equations

In hydrogeology, one of the most interesting problems is tracking pollutants in the groundwater. We will consider the case of porous media, where the physical

model uses a system of equations: one for the flow governed by Darcy's law and the conservation of mass, and the other for the transport based on Fick's law[2, 6]. Our point is to compute the center of mass c_m for this system. The equation for the flow is an Elliptic Partial Differential Equation (EPDE):

$$\begin{cases} v(x) & = K(x)\nabla p(x) \quad ,\forall x \in D \\ \nabla v(x) & = 0 \quad ,\forall x \in D. \end{cases}$$

where D is the domain, p is the hydraulic head, and K is the conductivity field. The solution of such an EPDE is called the velocity field in hydrogeology. The second equation is a PPDE from the family of Fokker-Planck Equations and possesses non constant coefficients. In fact, the drift coefficient is the solution to the above EPDE. The PPDE is the following:

$$\frac{\partial}{\partial t}c(x,t) + \nabla(v(x)c(x,t)) + \mathrm{div}(\sigma\nabla c(t,x)), \quad \forall x \in D, t \in [0,T],$$

with an initial condition $c_0(x)$ and a Dirichlet boundary condition on D.

The idea of Hybrid Simulation (HS) using Eulerian and Lagrangian techniques has appeared recently [3]. For the above system, HS has two steps:

1. solve the EPDE with an Eulerian Simulation and get the velocity field.
2. solve the PPDE using the velocity field and compute the center of mass.

As we are interested by the parallelization of MCS for the PPDE, we will suppose that the velocity field is computed in parallel using a mixed hybrid finite element and a sparse linear solver like in [3]. Such a scheme for the EPDE respects physical properties like conservation of mass while at the same time remaining highly parallelizable with a fast computation time. Since simulations of this model are very interesting for hydrogeologists on very large domains, we will need to introduce a domain decomposition method, otherwise we would not have enough memory. As a result, the velocity field is not stored on the memory of a single MP, and the scheme presented in section 3 can not be applied.

A solution proposed in [3] is to form what we can call a pipeline. Each subdomain is assigned to one MP which computes the path inside the subdomain owned by it. The easiest decomposition is the one-dimensional block column distribution, where each subdomain has at most its two nearest neighbors. With such a scheme the computation of the path will move from one MP to another possibly many times. Since every jump needs a communication between two MPs which provokes a slowdown, the decomposition should be as coarse as possible.

In [3], the convection term is dominant and the paths start from the first left column. Thus, they designed a strategy consisting of forming batches of paths run one after another to avoid having unused MPs. On the first MP, when a batch of paths is finished, every path which has not been absorbed is moved to the right column and new paths are launched. If we are on another MP, when a batch of paths is finished, we have a stack that is transmitted to each nearest neighbor causing the appearance of three cases:

1. on the right column, the stack becomes a new batch of paths to run,
2. on the left column, the stack is mixed with the one coming from its other neighbor,
3. the paths that hit the boundary are killed.

The above scheme has two major flaws: one is theoretical, the other is in the parallelization scheme. The former flaw is linked to the data decompostion, since the MPs only share their subdomains' boundary. As the result, the algorithm governing the motion at each timestep has to be modified. In fact, we need to be able to know if at a given timestep whether the path has left the subdomain or not. If it is, we have to compute a bridge of the underlying stochastic process to stop the path on the boundary and then adjust the length of the first timestep on its neighbor. We refer to [12] for more information on such a procedure. Regarding the second flaw, a problem is that some MPs finish earlier and lose load balance since paths will never come back due to a large convective force.

4.2 Dynamic Parallel Scheme

We present a new scheme based on the work introduced above and try to add MPs. We will show this scheme through the following example. Suppose the domain is divided on three MPs that we will call the Workers. At first, the additional MPs are part of a pool and named the Helpers. The first Worker is the one which contains leftmost division of the domain and starts the simulation by launching the first batch of paths. Since this Worker has all the planned paths to run, we provide it with Helpers from the pool which receive the Worker's data and start to run with a batch of paths too. Then, the first of the above MPs which has a path that is required to go to the next subdomain sends it to the Worker in the middle which starts to simulate immediately. This Worker will ask for new paths from the MPs which handle the first subdomain in order to continue simulating when it has finished. But these MPs might now have many paths to send, thus the second Worker inherits a full stack of paths to simulate. If this stack is too large, the receiver asks for help from the pool and provides batches of paths and the necessary subdomain to the Helpers. This procedure continues successively until every path is out of the domain.

A simpler scheme would consist of repeatedly cloning the three workers and running a batch of paths on each clone like in [3]. But, each clone clearly inherits a flaw mentioned above: the MPs who have finished might stay unused. Furthermore, this drastically increases the total number of MPs with poor load balance, since we do not have just three MPs but some multiple of three. In our scheme, we minimize this problem by letting the Workers and Helpers have the ability to rejoin the general process pool so that they may be reused elsewhere. But, this recycling principle raises three main problems:

1. When a Workers rejoins the pool and is reused, it loses its previous data. Thus, if a column of the domain does not remain on any MPs, we need to make sure that no paths have to revisit this subdomain. One solution is to ensure that the probability of coming back to this subdomain is very low before deleting the data. This clearly requires an incursion into the theory of rare events.
2. The amount of communication was very large in our test implementation, since we needed a master process to supervise the pool. Moreover, this master process is required in order to manage the end of the simulation. As a result, our implementation needs to either reduce the processing load on the Master process or to remove it entirely. Such a parallel scheme for MCS thus needs to borrow heavily from distributed algorithms literature.

3. The last problem is that we need to find the best use for the MPs from the pool. In fact, we distribute Helpers on demand for when there are too many paths to simulate. However, the computational complexity of a given set of paths may vary greatly. On one hand, the paths may be very quick to simulate if the advective force is large on some subdomains. On the other hand, there might be very few paths to simulate, but which take a lot of computational time. As such, the methods that are employed in allocating MPs should be adaptive and responsive to the demands of the process. This problem is at the forefront of artificial intelligence.

5 Conclusion

At the moment, the state of parallel Sparse Linear Solvers is much more advanced than that for Monte Carlo Simulation. However, inroads into this problem are beginning to foment. We are seeing that Stochastic Lagrangian Simulators are starting to give better results in parallel than Deterministic Lagrangian Simulators, and this in turn has created an avenue for parallelizing these types of simulators. An important and central subroutine that has lead to the advancement of Monte Carlo Simulators has been the underlying problem of Random Number Generation. This is still a current active topic of research. But, from our perspective, the possible improvements in this subject depends primarily on the management of the Virtual Random Number Generators by either attaching them to the Object, or to the ALU. We have shown that one should prefer the Strategy of Attachment to the Object. Either way, the same question arises in both strategies about how a complete parallelization of the hybrid simulation presented above can be achieved. As such, our future work will take place in the three fields mentioned in the previous subsection in order to realize more substantial computational performance.

6 acknowledgement

I start by thanking S. Maire and M. Simon who offer me the possibility to present this work at MCQMC. I thank J.Erhel and G.Pichot for the numerous discussions on Eulerian Methods. I am also grateful to T.Dufaud and L.-B. Nguenang for the instructive talks on the MPI library. C. Deltel and G. Andrade-Barroso of IRISA were of great help for the deployment on supercomputers and understanding the latest C++ standards. Many thanks to G. Landurein for his help in the implementation of PALMTREE. I am in debt to P. L'Ecuyer and B. Tuffin for the very interesting discussions about RNGStream. I show gratitute to D. Imberti for his help in the English language during the writing of this article. I finish with a big thanks to A. Lejay who taught me most of what I know about Stochastic Methods for PDEs. This work was partly funded by a grant from ANR (H2MNO4 project).

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] A. Beaudoin, J.-R. De Dreuzy, J. Erhel and G. Pichot. Convergence analysis of macrospreading in 3D heterogeneous porous media. *ESAIM: Proceedings*, 41:59–76, 2014

[3] A. Beaudoin, J.-R. De Dreuzy and J. Erhel. An efficient parallel particle tracker for advection-diffusion simulations in heterogeneous media. *Euro-Par 2007, LNCS 4641*, 705–714, 2007.

[4] 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.

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

[6] J.-R. De Dreuzy , A. Beaudoin and J. Erhel. Asymptotic dispersion in 2D heterogeneous porous media determined by parallel numerical simulations. *Water Resource Research*, 43, 2007.

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

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

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

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

[11] 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.

[12] A. Lejay and G. Pichot. Simulating diffusion processes in discontinuous media: a numerical scheme with constant time step. *Journal of Computational Physics*, 231:7299–7314, 2012.

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

[14] 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.

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

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

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

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

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