Stochastic control optimization and simulation applied to energy management: From 1-dimensional to N-dimensional problem distributions, on clusters and Blue Gene supercomputers

Stéphane Vialle [a] Xavier Warin [b] Constantinos Makassikis [a,c] Patrick Mercier [a]

[a] *SUPELEC, IMS research group, 2 rue Edouard Belin, 57070 Metz, France*

[b] *EDF - R&D, OSIRIS group, 1 Avenue Charles de Gaulle, 92141 Clamart, France*

[c] *LORIA, ALGORILLE project team, 54506 Vandoeuvre-lès-Nancy, France*

Abstract

This paper introduces the distribution of a stochastic control algorithm which is successively applied to a 1-dimensional problem: a gas storage valuation, and extended to a N-dimensional problem: an electricity asset management. This algorithm has been successfully implemented and experimented on PC clusters (up to 256 nodes and 512 cores) and on a Blue Gene/L and a Blue Gene/P supercomputers (using up to 8192 cores). Finally, the designed distribution allows to run gas storage valuation models which require considerable amounts of computational power and memory space, and to successfully optimize an electricity asset management problem with 7-energy-stocks and 10-state-variables while achieving both speedup and size-up.

Key words: stochastic control, message passing, multithreading, performances, PC clusters, Blue Gene, gas storage valuation, electricity asset management.
1991 MSC: 68W10, 68W15

Email address: `Stephane.Vialle@supelec.fr` (Stéphane Vialle).
URL: `http://www.metz.supelec.fr/metz/personnel/vialle` (Stéphane Vialle).

1 Motivations and objectives

Scientific objectives and locks: Since 1957, Dynamic Programming (see [4]) has been extensively used in the field of stochastic optimization. A lot of literature has been devoted to its use in reinforcement learning [14], in economy [15,9] and in finance specially for the pricing of options [6]. The success of this approach is due to the fact that its implementation by backward recursion is very easy. The main drawback of this method is due to the number of actions and the number of state control to test at each time step. The so-called curse of dimensionality brings a computational limit to this method. In order to tackle this problem some other methods have been derived. The stochastic dual dynamic programming has been used to optimize multi-stock hydraulic reservoirs [19] and decomposition methods have been used to treat so-called decomposable problems [8].

However, these methods need convexity of the underlying function to optimize or are not suitable for large multi-step optimization. As for the Stochastic Dynamic Programming method, it remains a general method useful for non-convex optimization in multi time step stochastic control problems but its utility is said to be limited to problems with less than 3 or 4 state variables involved. In the sequel, we show how to parallelize efficiently this algorithm when the goal of the optimization is to efficiently manage several stocks (energy, water) in order to save energetic resources and to maximize a profit on average in some asset management problems.

This research is part of the French national project "ANR-CIGC GCPMF" which gathers people from the academic world (computer scientists, mathematicians, ...) as well as people from the industry of energy and finance in order to provide concrete answers on the use of computational clusters, grids and supercomputers applied to problems of financial mathematics.

Project management and development strategy: This research project has started in March 2007, and the last experiments introduced in this article were achieved in May 2008. During these 15 months, the project has been split in two main parts:

- The first part was devoted to the parallelization of a 2-state control problem. We consider an asset manager who wants to maximize his profit by injecting and withdrawing gas from a gas storage plant when the price of gas is dependent on some uncertainties. This leads to a 1-dimensional problem with respect to the stocks (a single stock of gas is managed) where the two state variables are the stock in the cavity and the price of the gas. This case is interesting because it shows how our parallelization strategy can be used on a simple model, and the efficiency reached has encouraged us to apply our strategy to more difficult problems.

- The second part was devoted to a more complex problem: an electricity utility that wants to save energetic resources and optimize profit by managing *many* different stocks using thermal power plants in order to satisfy the load of electricity required by customers. In our example, the company wants to manage some stocks of water corresponding to some valleys and wants to take position in the future market. Each financial product can be seen as a stock of energy and adds a dimension to the problem treated. This kind of problem leads to a *N-dimensional* stock management under uncertainties due to the price of electricity, the load of customers and the inflows. The proposed solution is directly inspired from the solution to the problem of the first part. Experiments have allowed us to show how efficient our approach is and to postpone the current limit of the number of stocks and state variables that can be used by *Stochastic Dynamic Programming.*

From a parallel computing point of view, the computations involved in the two problems described above are both time and memory consuming. This makes large scale distributed architectures mandatory in order to achieve both speedup and size-up. However, these computations *can not* be parallelized in an *embarrassingly parallel* fashion and actually their parallelization involves many communications. As a result, the design of a complex parallel algorithm, including optimized communications and file management, was necessary in order to run our applications on large PC clusters (up to 512 cores) and on supercomputers (up to $8,192$ cores on a IBM Blue Gene/L and up to $32,768$ cores on a IBM Blue Gene/P).

Finally, in order to facilitate code maintenance, a unique version, running on both PC clusters and Blue Gene supercomputers, has been developed for each application. This has been achieved using (1) well known scientific libraries such as Blitz++ [21], Boost [1] and SPRNG [13]; (2) different implementations of the standard MPI library [16] for distributed computing; and (3) standard multithreading tools like OpenMP [17] or the Intel Thread Building Blocks library (TBB) [2].

Industrial objectives: The parallelization of the first software to valuate gas storage had two goals:

(1) The first was to derive an algorithm suitable for every kind of model used for the valuation. Currently used models are *one-factor Gaussian* models that do not need to be parallelized with complex algorithms: a multithreaded approach should be effective enough to get fast execution times. However, when testing some new models (with jumps for example) the time needed for a single run becomes crippling. This makes paral-

lelization essential in order to catch the gain of new models in reasonable times. Hence, the purpose of the parallelization was to provide a useful tool for researchers.

(2) The second was to prepare the path to parallelize the second problem which is at the heart of EDF's activity: the *electricity asset management.*

At EDF, due to the size of the problem, the electricity asset management is divided into three parts:

(1) The first part is dedicated to the management of the nuclear assets: it is used to decide when to stop the nuclear power plants for maintenance and refueling. This is the long-term unit commitment.
(2) The second part uses decisions taken into the first part to optimize the profit by managing the stocks of water, constraints on emission, future contracts, and other customer contracts. Some stocks of water and customer contracts are aggregated in order to solve the problem. This is the mid-term unit commitment.
(3) The last part decides which decisions to take for the next two weeks. All the physical constraints are taken into account, each hydraulic valley is accurately described and the final values of the stocks of water are given by the mid-term unit commitment. This part is called the short-term unit commitment.

The application which is parallelized in this paper deals with the mid-term unit commitment. The goal of the parallelization is to prepare the replacement of a current software used in production that uses heuristics to optimize the stocks. The portfolio is hedged every week but a lot of sensitivity calculations have to be carried out before each action as future purchase for example. In order to achieve all these studies meaning a lot of optimizations and simulations, *one optimization followed by a simulation part have to be achieved in less than two hours.* Our distributed software will also permit us to test some modelings to simplify the problem by giving reference calculation. In two or three years it could be used in production too.

Plan of the article: The first part of this article, which starts at section 2, introduces the distribution of the 1-dimensional stochastic control algorithm applied to a gas storage valuation. Subsection 2.1 describes the mathematical problem and the chosen algorithm to solve it, subsection 2.2 introduces a distribution of this algorithm, subsection 2.3 gives some details about the implementation and the development process, and subsections 2.4 and 2.5 show the achieved experimental performances. The second part of the article starts at section 3 and introduces the distribution of a N-dimensional stochastic control algorithm applied to energy asset management. The four subsections 3.1 to 3.4 introduce the mathematical problem, the designed distributed algorithm, the implementation process and the experimental performances we achieved on PC clusters and Blue Gene supercomputers. Subsections 3.5 and 3.6 focus on the amount of output data the application has to store in parallel from different computing nodes, and on the performances of our dynamic provisioning mechanism. Finally, section 4 summarizes the results and performances that were obtained before introducing the perspectives from both a parallel computing and application point of view.

2 Distribution of a 1-dimensional stochastic control algorithm

2.1 Definition and resolution of a gas storage valuation

2.1.1 Description of the problem

A gas storage facility presents three regimes: injecting gas, withdrawing gas, and just storing the gas. The gas storage is a cavity characterized by:

- its size given in giga British Thermal Units (BTU) or MWh (a standard conversion rate is used to convert BTU to MWh preferred by electric utility);
- the daily injection/withdrawal capacity a_{in}/a_{out} which depends on the stock level of the cavity I_t;
- the standard operating and managing cost per day which depends on the operating regime: $K_{in}(I_t)$, $K_s(I_t)$, $K_{out}(I_t)$.

The storage size can be variable in time because we may want for example to hire a portion of the cavity.

Most of the time, the gas storage manager uses its facility according to a *bang bang strategy*. In this case, the instantaneous gain (or cost) at a date t depends on the gas price S_t and the management regime. Here are the characteristics of the three regimes:

$$\begin{cases} \text{Injection} & a_{in}(I_t), \text{ with cost: } \phi_{-1}(S_t, I_t) = -S_t a_{in}(I_t) - K_{in}(I_t) \\ \text{Storage} & \text{ with cost: } \phi_0(S_t, I_t) = -K_s(I_t) \\ \text{Withdrawal} & a_{out}(I_t), \text{ with gain: } \phi_1(S_t, I_t) = S_t a_{out}(I_t) - K_{out}(I_t) \end{cases}$$

The instantaneous evolution of the stock I_t depends on the regime of the facility:

$$\begin{cases} dI_t = a_{in}(I_t)dt & \text{in injection} \\ dI_t = 0 & \text{in storing} \\ dI_t = -a_{out}(I_t)dt & \text{in withdrawal} \end{cases}$$

We suppose that a strategy u_t describing the regime taken at date t can take three values: 1 in withdrawal regime, 0 in storing regime, -1 in injection regime. We suppose that the gas storage is hired between t and T, and that the regime switching can occur at any date. At last, for simplicity, we take a zero interest rate. Then the gain obtained by managing the facility from a date t to a date T with a strategy u is given by:

$$J(t,s,c,i,u) = \mathbb{E}(\int_t^T \phi_{u_r}(r,S_r,I_r)dr + J(T,S_T,I_T,i_T,u_T)|S_t = s, I_t = c, u_t = i) \quad (1)$$

where:

- $J(T,S_T,I_T,i_T,u_T)$ is a given final value function, for example a penalization of the difference between I_T and a target final value I_T^{target};
- s is the gas price at time t given by a Markovian process;
- c is the stock level at time t;
- i is the regime at time t.

The goal of the manager is to find an optimal admissible adapted strategy in a given set $\mathcal{U}_t$, in order to maximize its income and therefore to solve:

$$J^*(t,s,c,i) = \sup_{u \in \mathcal{U}_t} J(t,s,c,i,u) \quad (2)$$

2.1.2 Stochastic control algorithm

In our models, the price of electricity is given by a Markovian process. We use *stochastic dynamic programming* in order to optimize the management of the facility. The stock is discretized in equally spaced levels. Furthermore, the regime switching occurs only at given dates (once a day): so we discretize time in equally sized intervals Δt. From the final value of J^*, we evaluate the value J^* for all the previous dates and all the levels of the stock with the algorithm of figure 1.

This algorithm is a generic one: the price model only appears in the conditional expectation. The time loop (t variable) is inherently sequential, unlike the

```
For $t := (nbstep - 1)\Delta t$ to $0$
    For $c \in$ admissible stock levels
        For $s \in$ all possible price levels
            $\tilde{J}^*(s,c) := \max\left(-(a_{in}s + K_{in})\Delta t + \mathbb{E}\left(J^*(S_{t+\Delta t}, c + a_{in}\Delta t)|S_t = s\right),\right.$
                $+(a_{out}s - K_{out})\Delta t + \mathbb{E}\left(J^*(S_{t+\Delta t}, c - a_{out}\Delta t)|S_t = s\right),$
                $\left.-K_s\Delta t + \mathbb{E}\left(J^*(S_{t+\Delta t}, c)|S_t = s\right)\right)$
    $J^* := \tilde{J}^*$    //Set $J^*$ for the next time step
```

Fig. 1. 1-dimensional stochastic control algorithm (application to gas storage valuation).

stock level loop (c variable) which can be efficiently parallelized. However, some complex data exchange will be mandatory at the end of each time step (see section 2.2).

2.1.3 Conditional expectation algorithm

The previous generic algorithm used for stochastic control uses the calculation of the conditional expected gain associated with price uncertainties. In order to evaluate this expectation, different techniques are used:

- A trinomial tree is used to generate uncertainty factors for the Ornstein-Uhlenbeck processes [7]. With a *one-factor Gaussian model* a single tree is generated, so the expectation is evaluated very quickly (will lead to our "G" algorithm). This factor is characterized by two parameters: the short term volatility σ_S describing the diffusion cone of the uncertainty and the mean reverting term a_S that forces the uncertainty to remain not to far away from 0. With a *two-factor model* two trees are combined generating far more calculation (that will lead to our "G-2f" algorithm). Two more parameters are needed: the long term volatility σ_L an the long term mean reverting parameter a_L that describe the second tree. This second tree is incorporated in order to account for the long term move of the forward curve. Furthermore, the long-term tree has far more branches than the short-term tree due to a small value of the long-term mean-reverting coefficient. In the second model the memory needed explodes with the maturity of the evaluation.
- A Partial Integro Differential Equation is used to calculate the expectation in the third model using a Normal Inverse Gaussian Levy process with pure jumps [3] instead of a Gaussian process:

$$\frac{\partial f}{\partial t} - \int\limits_{\mathcal{R}} \left(f(x+y) - f(x) - \frac{\partial f}{\partial x}(x)y\right) K_{NIG(\alpha,\beta,\delta)}(y)dy - (\sigma_S\frac{\delta\beta}{\gamma} - a_S)\frac{\partial f}{\partial x} = S(x)$$

  where the kernel $K_{NIG(\alpha,\beta,\delta)}(y)$ behaves as $O(1/y^2)$ in 0 (it will lead to our "NIG" algorithm). This calculation is not very memory-demanding but is

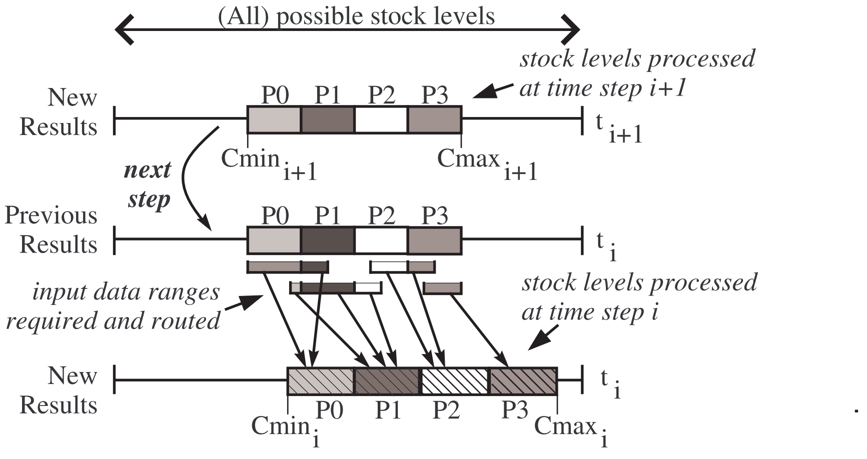
(All) possible stock levels
stock levels processed at time step i+1
New Results
P0 P1 P2 P3
t i+1
Cmin i+1
Cmax i+1
next step
Previous Results
P0 P1 P2 P3
t i
input data ranges required and routed
stock levels processed at time step i
New Results
t i
P0 P1 P2 P3
Cmin i
Cmax i

Fig. 2. Example of optimized data distribution on four processes.

far more costly than in the Gaussian model. Three more parameters are added to the one-factor Gaussian model in order to characterize this NIG model: α that is linked to the kurtosis of the distribution (thickness of the tail), β which is linked to the asymmetry of the distribution and δ which is a normalization factor a given function of β and α.

All details about the price models can be found in [22].

2.2 Optimized distributed algorithm

2.2.1 Distribution strategy

Modern *processors* include several *cores*, and *processes* can support several *threads*. Depending on the architecture and our implementation strategy, we can map one process per core, or one process per node and create threads to parallelize the process task treatment on the different cores (of the node). In the rest of this paper we consider *processes* independently of their physical mapping, and we detail their mapping and the complementary usage of *threads* when we introduce benchmarks and performance results.

To achieve large speedup and size-up, we have decided to parallelize the stochastic control algorithm of figure 1 on scalable distributed architectures, such as PC clusters and distributed memory supercomputers. Temporal steps of the external loop have to be run sequentially, but computations of the second loop on stock levels can be run concurrently. So, we have split the stock level loop on a set of processes communicating by message passing. However, the range of stock levels to process changes at each temporal step, and leads to redistribute computations and data at each step.

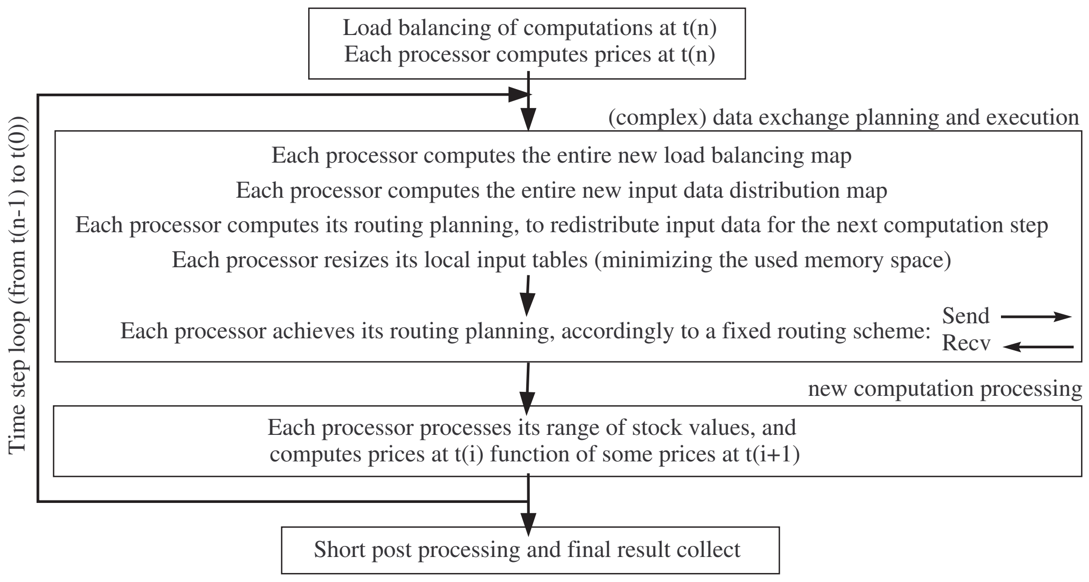
Load balancing of computations at t(n)
Each processor computes prices at t(n)
(complex) data exchange planning and execution
Each processor computes the entire new load balancing map
Each processor computes the entire new input data distribution map
Each processor computes its routing planning, to redistribute input data for the next computation step
Each processor resizes its local input tables (minimizing the used memory space)
Each processor achieves its routing planning, accordingly to a fixed routing scheme:
Send
Recv
new computation processing
Each processor processes its range of stock values, and computes prices at t(i) function of some prices at t(i+1)
Short post processing and final result collect
Time step loop (from t(n-1) to t(0))

Fig. 3. Main steps and sub-steps of our distributed algorithm for 1-dimensional stochastic control (applied to gas storage valuation).

As illustrated on figure 2, the stock level loop (from $Cmin_{i+1}$ to $Cmax_{i+1}$) is load balanced on processes at step $t = i+1$, and each process stores its results. At step $t = i$ (the next step), the new stock level loop (from $Cmin_i$ to $Cmax_i$) is load balanced on processes, and each process requires a specific range of the previous results as input data. Hence, each process computes the new input data range required to treat each stock level, and deduces the input data ranges required by all other processes to treat their new ranges of stock levels (according to the distribution of the entire new range of stock levels to process). Then, each process establishes its routing plan: it points out the ranges of its previous results to send to each other process, and the range of previous results to receive from each other process. Finally, each process executes its routing plan according to a predetermined message passing scheme, aiming to overlap the maximal amount of communications the hardware supports.

To store some new input data tables with different sizes at each step, some data tables are allocated and freed at each step in each process memory space. This memory management strategy introduces some small overhead, but in exchange minimizes the amount of memory used and therefore allows larger problems, requiring more processes, to be treated.

2.2.2 Main algorithm steps

According to the strategy introduced in the previous subsection, we have designed the distributed algorithm depicted in figure 3. This algorithm has three main steps, and the second step is the *time step loop* that includes two sub-steps. We continue to consider *processes* independently of their mapping on nodes and cores, and we detail the most important parts of this algorithm.

The first step consists in load balancing the computations of the prices at t_n by calling specific initialization routines. Then the algorithm enters a loop of n steps (from t_{n-1} to t_0) which encompasses two main sub-steps: *data exchange planning and execution*, and *new computation processing* (see figure 3).

The *data exchange planning and execution* sub-step starts computing the new load balancing map and the new input data distribution map on each process. These computations are simple, and are faster to compute entirely on each process than to distribute and parallelize. Then each process builds its routing plan and resizes its local data tables, according to the new input data distribution map. The data exchanges are achieved just after these preliminary operations, and are based on point-to-point communications.

The *new computation processing* sub-step is illustrated at the bottom of the time step loop of figure 3. It consists in a pure local and efficient computation on each process, according to the stochastic control algorithm of figure 1, and to a fixed price model.

2.3 Implementation and code test

Our first implementation was based on a Python top-level program calling MPI communication routines through the Python *Pypar* module [12]. This allowed users to easily tune the top-level program and run different distributed computations, and we have improved the MPI interface of the *Pypar* module. However our first implementation was limited to the use of `MPI_Bsend()` routine, did not run on Blue Gene/L (which did not support Python), and our experiments were limited to a small 32 PC cluster.

Our second implementation has been entirely achieved using MPI and C++ programming tools [11]. Three different versions have been implemented: using the blocking communication routine `MPI_Bsend()`, and the non-blocking routines `MPI_Ibsend()` and `MPI_Issend()`. Non-blocking versions allow each PC to parallelize and overlap its message sending and receiving at each step. Moreover, the non-blocking `MPI_Issend()` routine achieves non-blocking handshakes and requires a more complex design [16] but avoids to allocate extra communication buffers and to write out again data. As the required memory is less important, this implementation can reach greater size-up, when distributing large applications. Finally, its non-blocking handshake has exhibited very limited overheads in our application experiments, so we have used this communication routine to implement our communications.

2.4 Experimental performances

2.4.1 Distributed testbed features

Our distributed algorithms and implementations were assessed on three different testbeds. The first was the "Pentium-4 cluster" of SUPELEC which interconnects 32 PCs across a cheap Gigabit Ethernet network composed of two interconnected 24-port switches. Each PC has a Pentium-4 at 3 GHz and 2 GB of RAM. The second was the "dual Opteron cluster" of the French experimental Grid *Grid'5000*. It is composed of 72 nodes with two monocore Opteron processors at 2 GHz and 2 GB of RAM that are interconnected across a fast Gigabit Ethernet switch. The third machine was the IBM Blue Gene/L supercomputer of EDF R&D. It provides up to 4096 nodes and a total of 8192 cores, which communicate through proprietary high-speed networks. Each node hosts one dual-core PowerPC 440 processor at 700 MHz, and the 2 cores share 1 GB of RAM.

2.4.2 Benchmark application features

For the purpose of testing the application with our different price models described in subsection 2.1.3 we consider the following scenario where a gas storage owner wants to valuate his utility which has a capacity of $100,000$ MWh for a use during two years. The injection and withdrawal rates have values ranging between 100 and $1,000$ MWh per day and are highly dependent on the stock level. When the stock level in the cavity is high, the pressure is high too and makes injections more difficult than withdrawals. Conversely, when the stock level is low, injections are easier than withdrawals. The storage is valuated for a use beginning in one year and finishing two years later. The initial stock level is $20,000$ MWh and the final value of the gas storage in three years is set to 0 for simplicity. An annual interest rate of 8% is used as well as the forward prices available at Zeebruge hub at the beginning of 2006. The discretization step of the gas storage is set to 500 MWh.

The three stochastic price models are characterized by a daily short-term volatility equal to 0.014 associated to a daily short-term mean-reverting value of 0.0022 that totally define the first Gaussian model. The two-factor model needs two additional parameters to be defined: the daily long-term volatility set to 0.004, and the daily long-term mean-reverting set to 0.01. As for the Normal Inverse Gaussian model it also needs two more parameters: the first one α is set to 0.5 and is related to the kurtosis of the distribution while the second one β is set to 0 and is associated to the asymmetry of the distribution. The time step used for the three methods is 0.125 day. Such a refined time step is not necessary for the valuation itself, but it is to calculate the optimal

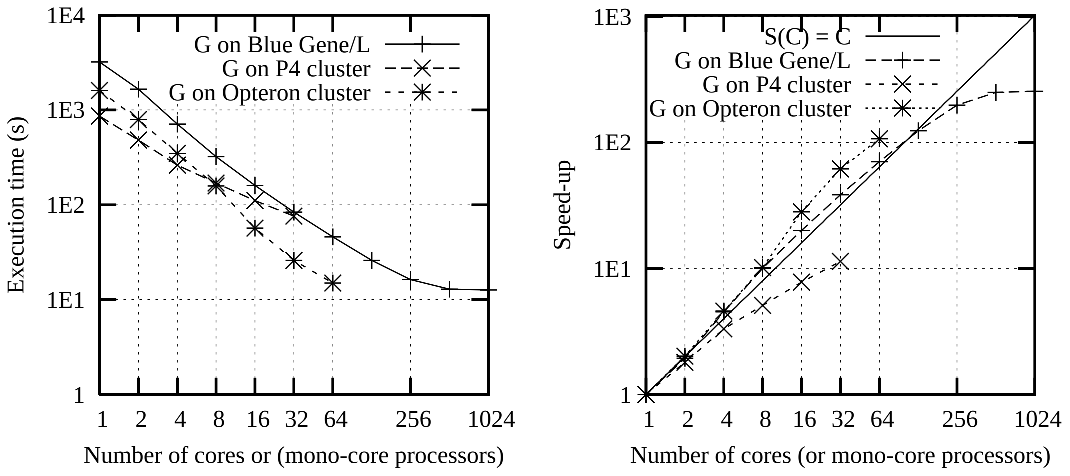

Fig. 4. Execution times (in logarithmic scale) and speedups of the Gaussian algorithm on three different distributed architectures, using only one core (or one mono-core processor) per node.

command that could be used by a Monte Carlo simulator in order to get, for example, the cash distribution generated each month during the two years. The Normal Inverse Gaussian model also needs a step for the *space* discretization. This step is set to 0.0125.

Using the above configuration, our three models yield respectively $1,355,010$€, $1,358,930$€ and $1,354,630$€ which correspond to the renting price of our fictive gas storage space calculated for a two-year period. In this gas storage test case, it can be noticed that the prices obtained by the three models are nearly identical. Nevertheless, other tests have to be carried out in order to determine whether the sophisticated model can outperform the one-factor Gaussian model for different types of gas storage. As it is shown by the performance results in the following section, such an investigation is made possible by our distributed implementation.

2.4.3 Execution times and speedups

Gaussian algorithm: Performances of this algorithm are sensitive to the per-node number of cores that are used. When using two cores per node and half of the nodes, the Blue Gene/L performances do not deteriorate: using C cores on C nodes or $C/2$ nodes leads to the same execution time. At the opposite, the dual Opteron cluster performances fall considerably: it is slower to use C cores on $C/2$ nodes than on C nodes, and it is faster to use C cores on C nodes than to attempt to use $2.C$ cores on C nodes! So its better to use only one core per node on our PC clusters and two cores per nodes on the Blue Gene/L to run the distributed *Gaussian* model. Performance measures are summarized on figure 4. When using only one core per node, the Blue

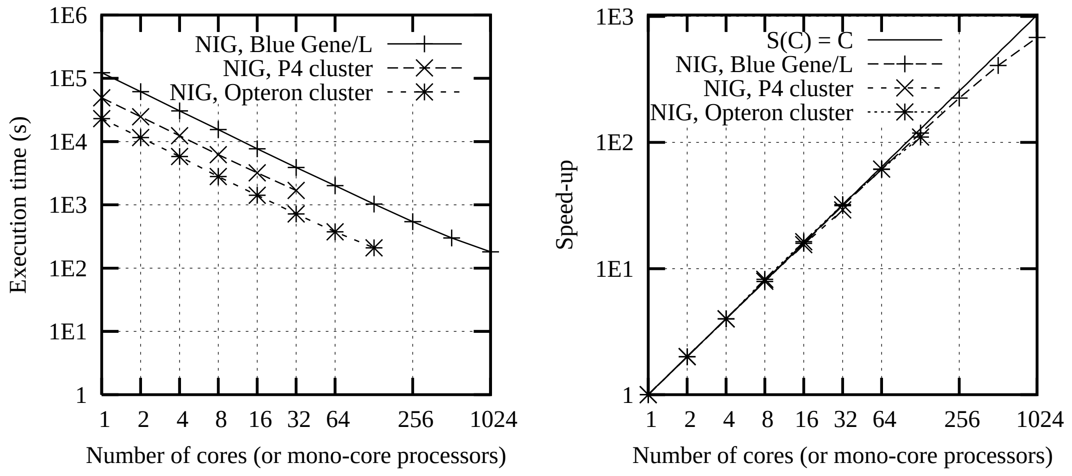

Fig. 5. Execution times and speedup of the Normal Inverse Gaussian algorithm on three different distributed architectures, using the maximum number of cores (or mono-core processors) per node.

Gene/L supercomputer and the dual Opteron cluster achieve a superlinear speedup from 4 to 64 cores. This can be explained by the improved cache performance obtained from the smaller memory requirements per node as a result of the distribution of the data. However, this superlinear speedup tends to disappear, and the Blue Gene/L speedup reaches its maximum at 512 cores. Even on a supercomputer our parallelization of the *Gaussian* algorithm does not scale beyond 512 cores, and its performance surpasses just a little bit the one on our high-end dual Opteron cluster using 64 cores (on 64 nodes). As for the cheap Pentium-4 PC cluster, it does not achieve any superlinear speedup and has a slowly increasing speedup curve. So, a fast interconnection network seems mandatory to achieve good performances on this distributed application, but a medium size mono-processor PC cluster with one core per processor and a good Gigabit-Ethernet switch can be a sufficient solution to run this distributed *Gaussian* algorithm.

Finally, despite some scalability problems that have been encountered, the best sequential execution time which was close to 15 minutes has been successfully decreased to 13 s - 15 s on a high-end cluster and a Blue Gene/L supercomputer. This is a real improvement for users, that frequently run this reference algorithm.

Normal Inverse Gaussian algorithm: Our distributed *Normal Inverse Gaussian* algorithm and implementation have reached very good performances independently of the per-node number of cores that was used. Figure 5 introduces performances achieved using the maximum number of cores per node on the Pentium-4 cluster, the dual Opteron cluster and the Blue Gene/L supercomputer. The performance curves exhibit an almost perfect parallelization

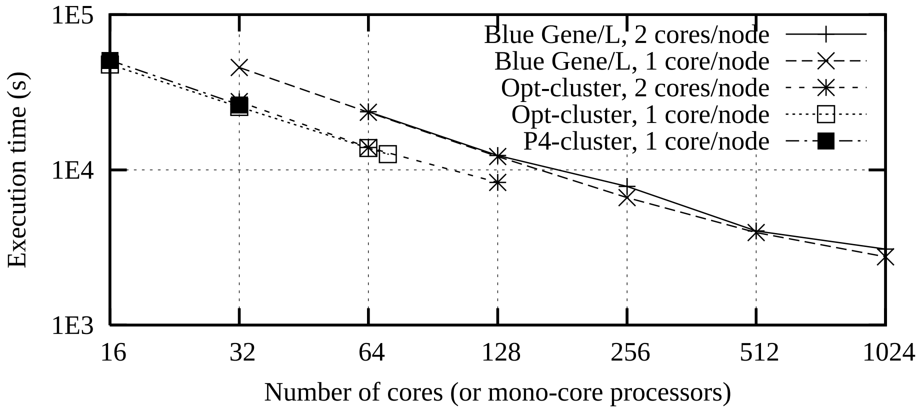

Fig. 6. Execution times of 2-factor Gaussian algorithm on 3 different architectures, using 1 or 2 cores (or mono-core processors) per node.

of the *Normal Inverse Gaussian* algorithm on all these architectures, even on the *cheap* Gigabit-Ethernet Pentium-4 cluster. The lowest execution time is achieved by the Blue Gene/L when using 1024 cores. However, the dual Opteron PC cluster achieves similar performances with only 128 cores. Hence, a large PC cluster with powerful multi-core nodes can be an interesting alternative to run our distributed NIG algorithm, and this regardless of its interconnection network.

In the end, our best sequential execution time which nears 6h25 (obtained by a mono-core Opteron processor) has been decreased to 3 minutes using 1024 cores of Blue Gene/L. Thus, our distribution makes it possible for users to exploit the *Normal Inverse Gaussian* algorithm provided they can mobilize enough computing resources.

2-factor Gaussian algorithm: The distributed *2-factor Gaussian* algorithm requires both huge amount of CPU and memory. With the current set of parameters (see section 2.4.2) and our implementation which mainly uses two tables - for storing the old and the new results - the application would theoretically require $2 \times 5,895$ MB of memory to execute sequentially. Hence, the application would easily be run on 8 nodes equipped with 2 *GB* of RAM, running a total of 8 MPI processes and requiring $1,474$ MB per MPI process. However, due to the nature of the stochastic algorithm and our distribution strategy, the overall memory needed when parallelizing is greater than in the sequential case. Furthermore, the kernel of the host operating system as well as the presence of communications which are handled by MPI contribute to increase the memory use. As a result, in practice, the minimum requirement to run this algorithm without swapping is 10 nodes with 2 GB of memory each, hosting a total of 10 MPI processes.

Figure 6 shows the execution times measured. The small 32 Pentium-4 PC

Table 1
Total amount of communications per step, function of the price model used and the number of MPI processes run

Price model	G	NIG	G-2f
Executable on 1 proc. with 2 GB of RAM	*yes*	*yes*	*no*
Total amount of communications per step on 32 proc. (with 32 MPI processes)	1.15 MB/step	0.99 MB/step	341.12 MB/step
Total amount of communications per step on 64 proc. (with 64 MPI processes)	2.34 MB/step	2.01 MB/step	699.42 MB/step
Total amount of communications per step on 128 proc. (with 128 MPI processes)	4.72 MB/step	4.05 MB/step	1399.08 MB/step
Total amount of communications per step on 256 proc. (with 256 MPI processes)	9.42 MB/step	8.11 MB/step	2806.24 MB/step

cluster has been able to run this benchmark from 16 to 32 mono-core processors, with 2 GB of memory per node (and per core). On 32 cores the execution time was approximately half of the execution time on 16 cores: the G-2f application seems to scale on this basic cluster. The dual Opteron cluster, which is equipped with 2 GB of memory per node, could also run our experiments with 16 nodes. Execution times on C cores were approximately the same on C nodes and on $C/2$ nodes, and the benchmark successfully scaled up to 128 cores using 64 nodes and 2 cores per node. Finally, on Blue Gene/L, with only 1 GB of memory per node, 32 nodes and their memories were required. Execution times were a little bit longer when using C cores on $C/2$ nodes instead of C nodes, but the slow down was not so important. Hence, like on the dual Opteron cluster, the G-2f application has also been run on the Blue Gene/L using two cores per node. Figure 6 shows that the G-2f application scales very well up to 128 cores on the dual Opteron cluster and up to 1024 cores on the Blue Gene/L machine. Finally, the Blue Gene/L machine appears to be the most interesting system to run the long G-2f application.

Our distributed *2-factor Gaussian* algorithm has succeeded in making possible these simulations (using the memory of at least 16 or 32 nodes), and has yielded results in 46 minutes on 1024 cores. However, the computation of the usual speedup is impossible since the G-2f benchmark could not be run in the memory space of a single node (of our testbeds).

Table 2
Cores (or mono-core processors) required to achieve G-2f test in 12,000 s, function of the accuracy.

q-discr factor:	2000 MWh	1000 MWh	500 MWh	250 MWh
Simulation accuracy:	rough simulations			fine simulations
bi-Opteron cluster, 2 proc./node	18 proc.	38 proc.	88 proc.	-
Blue Gene/L, 1 proc./node	32 proc.	63 proc.	132 proc.	280 proc.
Blue Gene/L, 2 proc./node	32 proc.	63 proc.	132 proc.	286 proc.
Simulation result (pricing result)	$11,065$ €	$13,052$ €	$13,589$ €	$13,870$ €

2.4.4 Measure of the total amount of communications

We have measured the total amount of data exchanged per step across all MPI processes by our application, function of the price model and the number of MPI processes. Table 1 shows these amount of communications:

- The "G" and "NIG" price models lead to different computations but to similar data accesses and communications. At the opposite the "G-2f" model considers greater amount of data and generates much more communications.
- The influence area of a stock point is independent of the number of processes, and due to our distribution strategy (splitting the range of stock points to visit) the overflow of the influence area of one process on its neighbors remains almost constant. Table 1 illustrates this mechanism, the total amount of communications has approximately a linear increase with the number of MPI processes, for all price models.

Finally, when using a large number of nodes and running a large number of MPI processes, the total amount of communications per step becomes huge, specially with the "G-2f" price model. When using thousands of nodes, this application makes full use of the fast interconnection network of the Blue Gene architecture.

2.5 Scalability experiment

Table 2 details the number of cores required to run the G-2f benchmark in 12,000 s for different simulation accuracies and on different system configurations. This extensibility experiment shows it is possible to maintain constant the execution time of the G-2f application when increasing its accuracy to get better results. The last row of table 2 shows that the results' values improve and that variations minimize when the discretization factor *q-discr* decreases. But the number of required cores tends to double when the discretization is twice finer. The Blue Gene architecture has been designed to *scale* up to a hundred thousands cores (to reach PetaFlops), and it is easy to mobilize the required number of cores to achieve a simulation with a 250 MWh discretization factor in $12,000$ s, while this was not possible on the dual Opteron cluster. Moreover, table 2 shows that 286 Blue Gene/L cores instead of 280 are required to run a strongly accurate simulation in $12,000$ s when using two cores per node instead of one. As this overhead is small, it is better to use C cores on $C/2$ nodes since it mobilizes less computing resources.

Finally, this scalability benchmark shows that our distributed strategy and implementation successfully *scale* on a Blue Gene/L as well as on a PC-cluster architecture, when we aim to increase the simulation accuracy.

3 Distribution of a N-dimensional stochastic control and simulation

3.1 Definition and resolution of an electricity asset management problem

In this section we detail a specific problem we want to solve, leading to compute a N-dimensional stochastic control and simulation and to extend the 1-dimensional algorithm introduced in section 2.2. We consider the situation of a power utility that has to satisfy customer load, using the power plants and hydro reservoirs at its disposal. The utility also disposes of a trading entity being able to take positions on both the spot market and futures market. Finally, uncertainty will play a key role in the problem and has to be taken into account as well. The model is fully developed in [23].

3.1.1 Problem definition

We will use a *one-factor Gaussian model* for load. In this model, load will randomly evolve around an average load. The load is noted $D(t)$. The price model for the future curve is a two-dimensional Brownian motion. We note $F(t,p)$ the stochastic price at time t for the product p defining the delivery of one MWh during one month $[t_b(p), t_e(p)]$.

We introduce some notation for our market products:

$\mathcal{P}(t) = \{p : t < t_b(p)\}$	all futures with delivery after t,
$L(t,p) = \{\tau : \tau < t, p \in \mathcal{P}(\tau)\}$	all time steps τ before t for which the future p product is available on the market,
$\mathcal{P}^t = \{p : t \in [t_b(p), t_e(p)]\}$	all products in delivery at t,
$\mathcal{P} = \cup_{t\in[0,T]}\mathcal{P}(t)$	all futures products considered.

Now we can write the problem to be solved as the minimization of the average cost of managing *nbunit* energy production units under constraints:

$$\min \mathbb{E}\left(\sum_{t=0}^{T}[\sum_{i=1}^{nbunit} c_{i,t}u_{i,t} - v_t S_t + \sum_{p\in\mathcal{P}(t)} (t_e(p) - t_b(p))q(t,p)F(t,p)]\right) \quad (3)$$

$$\begin{aligned}
\text{so that: } D_t &= \sum_{i=1}^{npal} u_{i,t} - v_t + \sum_{i=1}^{nres} w_{i,t} + \sum_{p\in\mathcal{P}^t}\sum_{s\in L(t,p)} q(s,p) \\
& R^i_{t+1} = R^i_t + \Delta t(-w_{i,t} + A^i_t) \ \ \forall i \in \{1,...,nres\} \\
& R^i_{min} \le R^i_t \le R^i_{max} \\
& q_{p,min} \le q(s,p) \le q_{p,max} \ \ \forall s \in [0,T] \ \forall p \in \mathcal{P} \\
& y_{p,min} \le \sum_{s=0}^{\tau} q(s,p) \le y_{p,max} \ \ \forall \tau < t_b(p) \ \ \forall p \in \mathcal{P} \\
& v_{t,min} \le v_t \le v_{t,max} \\
& 0 \le u_{i,t} \le u_{i,t,max}
\end{aligned}$$

Where:

- D_t is the customer load at time t in MW
- $u_{i,t}$ is the production of unit i at time t in MW
- v_t is spot transactions in MW (counted positive for sales)
- $q(t,p)$ is the power of the futures product p bought at time t in MW
- R^i_t is the level of the ith reservoir at time t in MWh
- S_t is the spot price in euros/MWh
- $F(t,p)$ is the futures price of the product p at time t in euros/MWh
- $w_{i,t}$ is production of reservoir i at time t in MW
- A^i_t are the reservoirs inflows in MW
- Δt the time step in hours
- $q_{p,min}$, $q_{p,max}$ are bounds on what can be bought and sold per time step on the futures market in MW
- $y_{p,min}$, $y_{p,max}$ are the bounds on the size of the portfolio for futures product p (market depth)
- R^i_{min}, R^i_{max} are (natural) bounds on the energy each reservoir can contain.

3.1.2 Problem resolution

An approach based on the use of *Stochastic Dual Dynamic Programming* has been proposed by Christiansen [5] to solve a similar problem. Due to the non-convexity of the formulation, hyperplane cuts have to be adapted coupling the classical approach to prices discretization. The convergence of the procedure is not very obvious and is even more difficult to extend to other kind of contracts (clearing contracts, for example, meaning *mixed integer programming problematic*). So, we have decided to use *Stochastic Dynamic Programming* that does not need convexity of the problem. This method only requires the price and the load model to be Markovian in order to be easily implemented. However, this approach is exponentially expensive with the number of stocks used (number of reservoirs and futures) and is highly CPU consuming.

We suppose that we want to optimize the production from now until date T with a time step Δt. We note $nbstep = T/\Delta t$ the number of steps used for the optimization. The resolution algorithm is a time backward algorithm given in figure 7 where $\phi(nc)$ is the cost associated to command nc, and s^* is the price and demand scenario at date $t+1$ following a scenario realization s at date t. In the algorithm of figure 7, at date t, $J^*(s,c)$ is the minimum cost for the asset management from date t to date T starting from stock level c and with the price and demand scenario s. $\mathbb{E}\left(J^*(s^*,c)|s\right)$ is a short notation for the conditional expectation of J^* at stock point c knowing that at date t the scenario s has happened. Another originality of our approach is the use of the Longstaff-Schwartz method [10] or Tsitsiklis-Van Roy method [20] to compute the conditional expectations instead of the classical use of scenario trees. This method which is often used in mathematical finance is to our knowledge rarely used on discrete optimization problems. It can be noted that in [23] we have proposed some approximation methods to reduce the computation time and memory needed to deal with future stocks. The algorithm of figure 7 takes into account several energy stock levels, but exhibits similarities with the 1-dimensional algorithm of figure 1 used to compute a gas storage valuation (see section 2.1.2). These similarities will lead to close distribution strategies (see following sections).

During the resolution of the problem described by equation (3) in the previous section, the Bellman values of J^* are stored on disk and used in a simulation part. This simulation part draws some new trajectories by a Monte Carlo method and the different values of stock are computed using the Bellman values calculated during the optimization part for each realization of the trajectories.

```
For $t := (nbstep - 1)\Delta t\ to\ 0$
    For $c \in$ admissible stock levels ($nbstate$ levels)
        For $s \in$ all possible price and demand scenario ($nbtrajectory$)
            $\tilde{J}^*(s, c) = \infty$
            For $nc \in$ all possible commands for stocks ($nbcommand$)
                $\tilde{J}^*(s, c) = \min(\tilde{J}^*(s, c), \phi(nc) + \mathbb{E}\left(J^*(s^*, c + nc)|s\right))$
    $J^* := \tilde{J}^*$        $//Set\ J^*\ for\ the\ next\ time\ step$
```

Fig. 7. N-dimensional stochastic control algorithm (application to electricity asset management).

3.2 Optimized distributed algorithm

3.2.1 Distributed strategy and application structure

The entire application is composed of three different parts which use parallel algorithms (see figure 8) run by a set of processes distributed on machine nodes [18]. The first part consists in *reading many input files* in order to store data in each process memory space and to execute some initializations. The second part of the application is the *optimization* part that computes the optimal decision to be taken depending on the state (levels of stocks and realizations of aleas) by a backward time recursion summarized by the algorithm on figure 7. As in the distributed algorithm of gas storage valuation (see sections 2.2.1 and 2.2.2), processing of the external time step loop can not be parallelized, but each time step can be processed in parallel: we can split internal loops on admissible stock levels and possible prices. However, the amount of data and computation can change at each time step. So it is mandatory to redistribute the new computations and their required input data at the beginning of each time step to ensure a good load balancing. Moreover, a lot of intermediate results (the so-called Bellman values) have to be stored on disks by all processes at each step. The third part is devoted to the *simulation* process. Some Monte Carlo scenarios for aleas are calculated and some trajectories of stocks are computed using the optimal commands calculated in the *optimization* part. It looks like a loop of embarrassingly parallel computations. But these computations need data scattered in all the intermediate result files of the *optimization* part, leading to many communications between processes. Finally, at each time step of the *simulation* part, some other application results have to be saved in files from each process in order to derive some risk indicators.

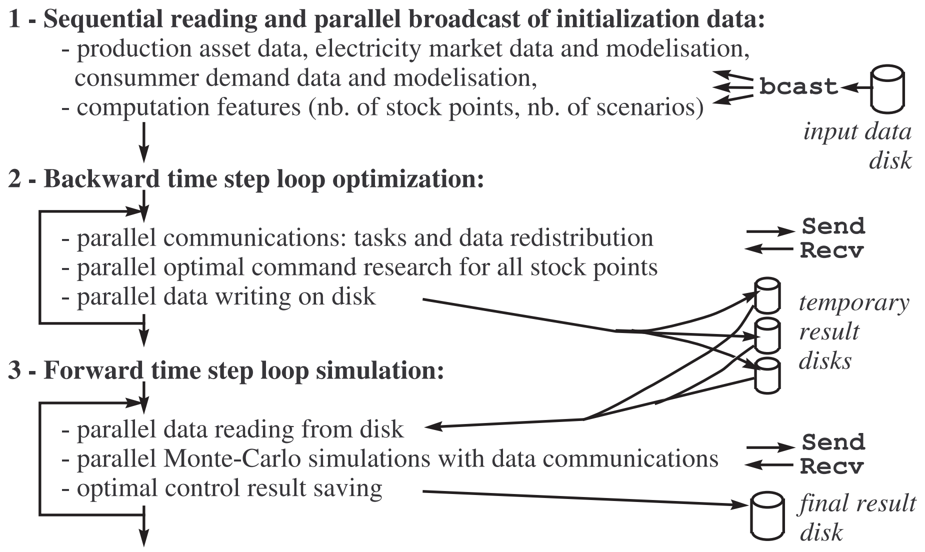
1 - Sequential reading and parallel broadcast of initialization data:
- production asset data, electricity market data and modelisation, consummer demand data and modelisation,
- computation features (nb. of stock points, nb. of scenarios)
bcast
input data disk
2 - Backward time step loop optimization:
- parallel communications: tasks and data redistribution
- parallel optimal command research for all stock points
- parallel data writing on disk
Send
Recv
temporary result disks
3 - Forward time step loop simulation:
- parallel data reading from disk
- parallel Monte-Carlo simulations with data communications
- optimal control result saving
Send
Recv
final result disk

Fig. 8. Main steps of our distributed algorithm for N-dimensional stochastic control and simulation (applied to electricity asset management).

3.2.2 Input data reading

Input data is stored in a set of files and has to be loaded in the memory spaces of all processes of our parallel application. Blue Gene/L supercomputer supports concurrent and straightforward file accesses, using its efficient parallel file system (GPFS) and IO mechanisms. However, our PC cluster does not support concurrent file accesses from hundred processes across NFS file system. So, we have implemented a more generic mechanism where process of rank 0 reads input files and broadcasts data to all other processes so that each process can make its own initializations. On both testbeds, when using this mechanism, the time needed to read data remains approximately constant when the number of nodes and processes increases, and is negligible compared to the rest of the application. It takes about $3s$ on our PC cluster and about $18.5s$ on Blue Gene/L supercomputer.

3.2.3 Stochastic control optimization

The number of used nodes is unconstrained on our PC cluster but has to be a power of two (2^d) on Blue Gene/L. So, to be generic when distributing the admissible stock level loop (see algorithm of figure 7), we have to split a N-dimensional cube of data and its associated calculations (named the *N-cube*) on an hypercube of $P = 2^k$ processes (considering again processes independently of their mapping on nodes and cores). However, due to the nature of data and associated computations, some dimensions of the N-cube do not have to be split. Finally, at each time step t_i, all processes run the splitting algo-

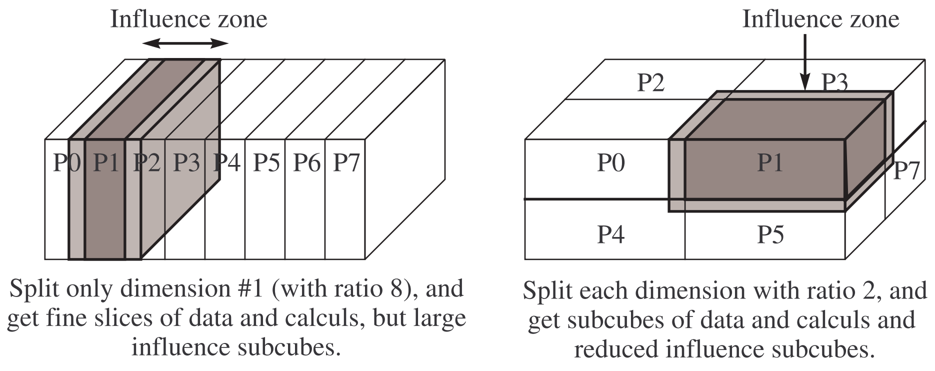
Influence zone
P0
P1
P2
P3
P4
P5
P6
P7
Split only dimension #1 (with ratio 8), and get fine slices of data and calculs, but large influence subcubes.
Influence zone
P2
P3
P0
P1
P7
P4
P5
Split each dimension with ratio 2, and get subcubes of data and calculs and reduced influence subcubes.

Fig. 9. Example of *flat* and *cubic* split of the N-Cube of data and computations

rithm to compute the entire map of data and computations distribution at t_i. Then a routing plan is computed and executed by each process to bring back in its memory space the required t_{i+1} input data to achieve calculations at t_i (this is a backward time step loop). Then, each node stores its *influence N-subcube* of t_{i+1} input data. As told before, this algorithm in an extension of the 1-dimensional algorithm introduced in section 2.2. To minimize the size of these *influence N-subcubes* and to achieve great size-up, our splitting algorithm attempts to create *cubic* sub-cubes of data and calculations on each node (in place of *flat* sub-cubes), see figure 9.

All communications of one routing plan can be overlapped and all data sizes are specified in the routing plan. Similarly to the 1-dimensional parallel algorithm (See section 2.3), the execution of the routing plan has been implemented with asynchronous communication routines that run in parallel in order to increase speedup and, in order to increase size-up, optimal in size data structures are allocated and communication routines that need not extra buffers to be allocated are used. Moreover, process computations are mainly composed of computing loops with independent iterations. These loops have therefore been multithreaded in a way where each thread manages a continuous subset of the iterations. As a result, by running one process and two threads per node on a dual-core distributed system architecture, it is possible to use all processing resources and to group communications on each node. However, this feature could only be experimented on a PC cluster since Blue Gene/L's operating system lacks support for multithreading. This limitation is expected to disappear in the Blue Gene/P.

Each process generates many intermediate results, and it is mandatory to flush these results on disk. Each process appends its intermediate local results to two data files, appends some global descriptions of the current problem step to two other files, and fills an *index file* to easily retrieve step data in the other files. On a PC cluster, each process fills 5 files on its local node disk, and a total of $5.P$ files are stored on local disks. On a Blue Gene/L supercomputer each process stores its intermediate results on a remote disk across an efficient parallel file system, and only process of rank 0 fills two files with global descriptions of each problem steps. So, only $2+3.P$ files are stored on one remote disk.

3.2.4 Monte Carlo simulations

The third part of the application consists in a set of N_s forward Monte Carlo (MC) simulations, and each process computes N_s/P simulations. These MC simulations are independent computations, but require to read the intermediate result files of the *optimization* part. In order to minimize file accesses we have implemented a forward time step loop, processing all MC simulations at each step. At step t_i, each process reads its intermediate result file of t_{i+1} *optimization* step, exchanges some t_{i+1} data with other processes and computes the evolution of its N_s/P simulations.

In fact, to progress from t_i to t_{i+1} each simulation requires data not always stored in the file of the process and communications are required to bring back an *influence area* in the process memory, like in the *optimization* part. A first solution is to enter a loop bringing back one influence area and processing one MC simulation at each step. A second solution is to compute and bring back one large influence area, including influence areas of the N_s/P MC simulations of the process, and then to enter a loop processing its N_s/P MC simulations. The first solution requires less memory but achieves more and smaller communications. However, it has exhibited quasi-insignificant overheads. So, by default we use the second solution (bring back one large influence area) to run benchmark (see 3.4.3), and we switch to the first solution when our application misses of memory space to run.

3.3 Implementation and code test

This electricity asset management application which mixes both message passing and multithreading paradigms in order to take maximal advantage of distributed architectures with SMP nodes (multiprocessor or multi-core nodes) has been implemented using the C++ programming language. Message passing was achieved using different MPI implementations (MPICH-1 and OpenMPI on PC cluster and IBM MPI on Blue Gene/L). As for multithreading it is available using either OpenMP [17] or the Thread Building Block Intel library (TBB) [2]. As in the implementation of the 1-dimensional stochastic control application (see section 2.3) non-blocking `MPI_Issend()` and `MPI_Irecv()` routines were used to overlap all communications of one routing plan execution, and some calls to `MPI_Wait()` synchronization routine were used to ensure that the communications are achieved before entering a new time step. The others scientific libraries that were used are Blitz++, Boost, SPRNG and Clapack. Finally, the same source code can be compiled both on Linux PC clusters and on IBM Blue Gene supercomputers, allowing to maintain only one code of approximately $65,000$ lines of C++.

The debug and validation of a distributed numerical application remains a hard part of the development process. Many errors do not lead to a complete crash of the program, but to a result slightly different of the correct one. The initial stochastic computing part of our code was a sequential code that had been previously validated. The debug and validation of the distributed version has been done in several steps:

- Memory management errors have been tracked using several debugging tools and check options of the Blitz++ library.
- Results of the distributed and sequential versions were compared.
- Intermediate result tables have been printed and analyzed in order to to track any value resulting from bad data exchange between processes but not leading to significant errors on our test applications.
- Several benchmarks have been run using different number of processes and different computing routines (leading to the same result).

So, the current distributed and parallel program has not been *proved*, but has successfully passed a strong validation process.

3.4 Experimental performances

3.4.1 Large scale cluster and supercomputer

We have experimented and evaluated our parallel and distributed implementation on three large distributed machines with small SMP nodes. The first machine was a 256-PC cluster of SUPELEC (from CARRI Systems company) with a total of 512 cores. Each node hosts one dual-core processor: INTEL Xeon-3075 at 2.66 GHz, with a front side bus at 1333 MHz. The two cores of each processor share 4 GB of RAM, and the interconnection network is a Gigabit Ethernet network built around a large and fast CISCO 6509 switch. The second machine was the IBM Blue Gene/L supercomputer of EDF R&D. It provides up to 4096 nodes and a total of 8192 cores, which communicate through proprietary high-speed networks. Each node hosts one dual-core PowerPC 440 processor at 700 MHz, and the 2 cores share 1 GB of RAM. The third machine was the new IBM Blue Gene/P supercomputer of EDF R&D, providing up to 8192 nodes and a total of $32,768$ cores. Each node hosts a quad core PowerPC 450 processor at 850 MHz, and the 4 cores share 2 GB of RAM.

On both machines it is possible to use all cores of each node to run computations, or to devote one core to manage communications. Since our application includes many communications, experiments were mandatory to point out the most efficient solution.

3.4.2 Experiment of a 7-stock management application with 10 state-variables

The starting day of the case is arbitrarily taken to be 01/08/2004 and the final date 30/11/2005. These two dates have no consequence whatsoever on what follows. The time step discretization is equal to one day. We consider a production portfolio consisting of 26 thermal units (so $nbunit = 26$) and one hydraulic unit. The benchmark we used is based on the problem described in equation (3) with 1 stock of water (so $nres = 1$ which corresponds to the management of the hydraulic unit) and 6 stocks of futures (so $\mathcal{P}(0) = 6$). We recall that we consider 3 Gaussian aleas: 1 for the customer load and 2 for the prices, and we consider peak and off peak futures with monthly delivery periods. Three periods of delivery (Sept., Oct., Nov.) each with peak and off peak delivery are considered leading to a 7 stock problem with 10 state variables (7 stocks + 3 aleas).

The depth of the market for the 6 future products is set to 2000 MW for purchase and sales (so $y_{p,min} = -2000$, $y_{p,max} = 2000$, see section 3.1). Every two weeks, the company is allowed to change its position in the futures market within the limits of market depth by either buying 1000 MW, doing nothing, or selling 1000 MW. The future stocks are discretized with a 1000 MW step and the water stock is discretized with a 20000 MWh step (some parameters have MWh units while others have MW and are associated to time periods). The hydraulic command is tested with a step of 1000 MW. All the commands for the futures stocks are tested from -2000 MW to 2000 MW (so $q_{p,min} = -2000$, $q_{p,max} = 2000$) with a step of 1000 MW.

This problem leads to the following parameters in the algorithm introduced in section 3.1.2 and figure 7: $nbstep = 121$; $nbstate = 225 * 5^6$; $nbcommand$ (the number of command to test for each point) is between 5 when non position in the future market is taken, and $5 * 3^6$ when all stocks of futures are available for purchase or sell; $nbtrajectory$ (the number of scenarios in optimization) is equal to 400. This discretization is a very accurate one leading to a huge problem to solve. The result of this optimization is 898.5 Millions Euros. It represents the optimal cost of management during these four months, and is coherent with the results obtained by some simplified approach.

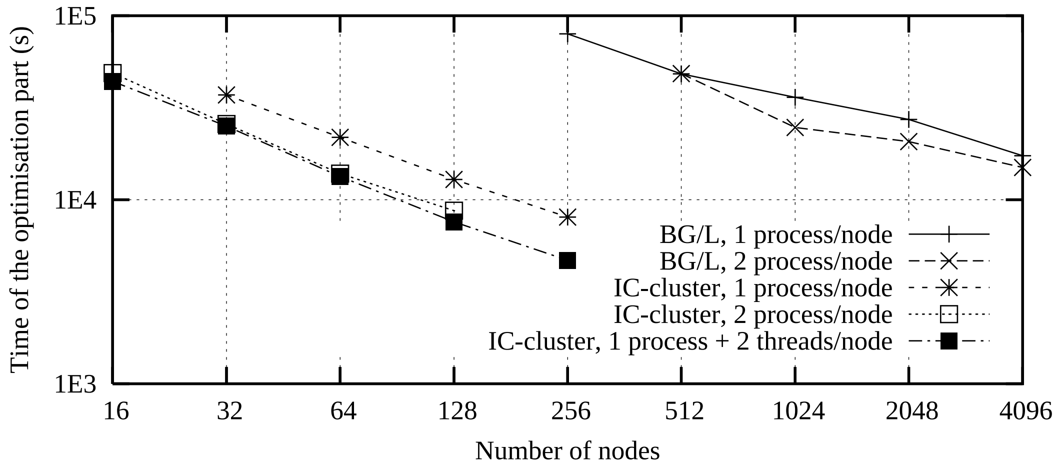

Fig. 10. Execution time of a 7-stock and 10-state-variable stochastic control optimization, using MPICH-1 and Intel TBB libraries on a PC cluster, and using IBM MPI library on a Blue Gene/L supercomputer.

3.4.3 Execution times and scalability

Speedup and size-up of the optimisation part: The most time and memory consuming part of the application is the *optimization* part, and our first experiments have focussed on this part. We have computed the optimisation of the 7-stock problem introduced in section 3.4.2 on our PC cluster and Blue Gene/L supercomputer testbeds, using one or two cores per node. Figure 10 exhibits the performance curves we measured. A global overview of these curves shows our distributed algorithm and implementations both speed up, size up, and scale. Our 7-stock problem requires too much memory to be processed on one node, but we succeeded in processing it using at least 16 nodes of our PC cluster or 256 nodes of our Blue Gene/L supercomputer. Then, using more nodes achieves regular increase of performances on both testbeds. Finally, when multiplying the number of nodes by 16, best implementations achieve a speedup of 9.36 on PC cluster and close to 5.31 on Blue Gene/L.

On our PC cluster it appears efficient to use two cores per node for our computations, and running one MPICH-1 process and two threads on each node appears a little bit more efficient than running two MPICH-1 processes. On our Blue Gene/L, a 1-stock version of this stochastic control algorithm used for gas storage valuation efficiently run on two cores per node (see section 2.4 and [11]). However, it appears more efficient to run this 7-stock version on two cores per node in the range $[1024 - 2048]$ nodes, while outside of this range the improvement seems weak. But performance measurements encountered many fluctuations on Blue Gene/L, and some measures on figure 10 are average values and seem (too) long. Some investigations are planned in collaboration with IBM to analyze and optimize the Blue Gene/L implementation.

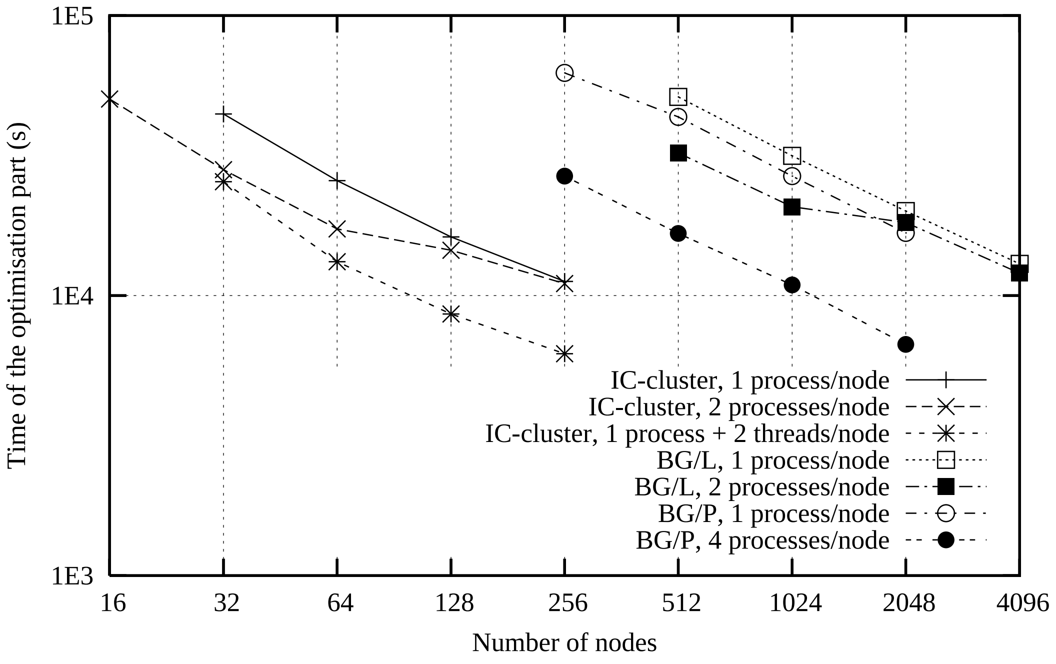

Fig. 11. Execution time of a complete 7-stock and 10-state-variable stochastic control (optimization and simulation), using OpenMPI and OpenMP libraries on a PC cluster, and using IBM MPI library on a Blue Gene/L supercomputer

Performance of the entire parallel and distributed application: The MPICH-1 library was running fast on our PC-cluster but has led to frequent failures beyond 64 processes. So, we have used OpenMPI library for next experiments. It has appeared a little bit slower but has removed failures.

First experiments of our entire distributed application (stochastic control *optimization* and *simulation*) on our 7-stock problem (using OpenMPI on our PC-cluster), have reached the performances illustrated on figure 11. These executions include many computations, communications and file IO operations. The *optimization* part (see section 3.2.3) remains the most important part and still exhibit a good scaling. But the *simulation* part (see section 3.2.4) does not scale as much as the *optimization* part, and becomes a more important fraction of the execution time when the number of nodes increases. Consequences are illustrated on the cluster performance curves of figure 11: the execution time decrease becomes less important when the number of nodes increases. Running two OpenMPI processes per node accentuates this phenomenon, decreasing the granularity of the communications and file IO. However, an interesting solution consists in using the two cores of each node running only one OpenMPI process and two threads per node. It increases the parallelization of the computations while maintaining the granularity of communications and file IO operations. It leads to a speedup close to 1.8 compared to the use of only one core per node.

Table 3
Size of intermediate and final result files (on a 256 node PC-cluster).

	On PE-0	Sum on all cluster nodes	
		mode *local*	mode *remote*
Intermediate result files	33.7MB	5.33 GB	4.85 GB
Final result files	807.2 MB	-	-

On the IBM Blue Gene/L supercomputer we can observe the performance curves have similar trends to the curves on PC-cluster. Up to 1024 nodes, using IBM MPI library and 2 MPI processes per node decreases the execution time and improves performances, but beyond 1024 nodes 2 MPI processes per node are not more efficient than only 1 MPI process per node (like on our PC-cluster). However, when using only 1 MPI process per node we get a very linear execution time curve, meaning this execution time decreases very regularly: Blue Gene/L architecture seems to scale very well on this benchmark (better than our PC-cluster). Unfortunately, it is not possible to implement multithreading on Blue Gene/L architecture and to compare to the MPI-OpenMP implementation on our PC-cluster.

On the new IBM Blue Gene/P of EDF company we have succeeded to run our benchmark using the IBM MPI library, and running 1 or 4 MPI processes per node. We can see the Blue Gene/P is a little bit faster when running only 1 MPI process per node than the Blue Gene/L, and seems to scale very well up to 2048 nodes and 2048 cores. When running 4 MPI processes per node in place of 1 the execution time is divided by 2.5 approximately, and the execution time decreases very regularly, down to 6700 s on 2048 nodes and 8192 cores. So, using the entire $32,768$-core Blue Gene/P and using both IBM MPI and IBM OpenMP libraries should lead to a serious improvement of this performance. However, we need to solve some technical installation problems before to make a full use of this supercomputer.

3.5 Output file sizes and storage mechanisms

The benchmark application introduced in section 3.4.2 generates important intermediate and final result files. Table 3 introduces the sizes of these output files.

The *intermediate* files are written by the *optimization* part and read by the *simulation* part. Section 3.2.3 introduces the two possible IO modes to store intermediate data: each MPI-process stores 5 files on its local disk(mode *local*), or each MPI-process stores 3 files on remote disks and process 0 store 2 additive files (mode *remote*). The *local* mode is the most efficient when each

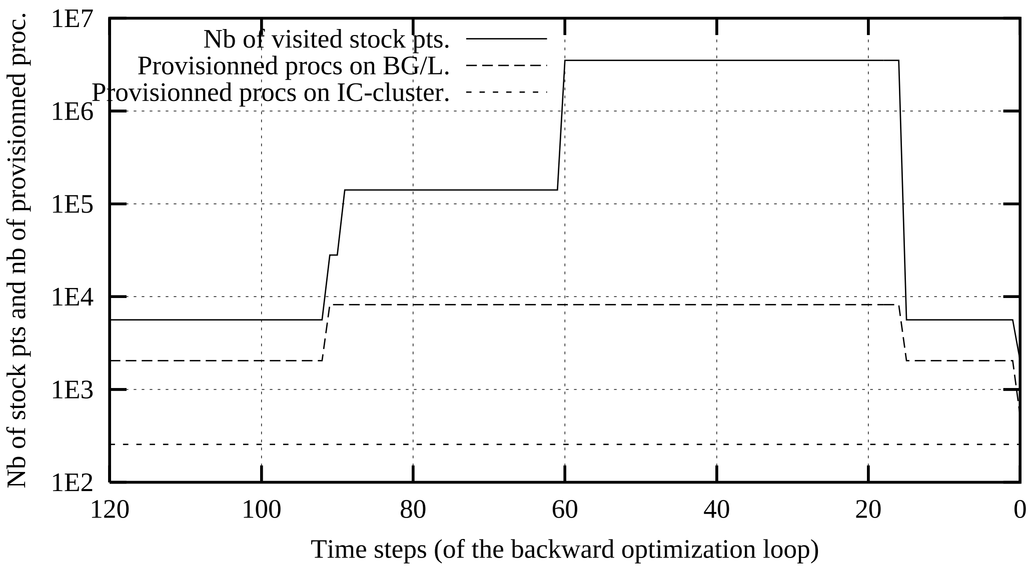

Fig. 12. Evolution of the problem size (nb of visited stock points) and of the number of provisioned MPI processes, running 8192 MPI processes on a Blue Gene/L and 256 MPI processes on a PC cluster.

node has a local disk (like on our PC-cluster) and does not require any specific file system. The *remote* mode is most adapted when all nodes have to access a remote disk space across an efficient file system (like on a Blue Gene architecture). The amount of intermediate results stored by one process on one node is not so large (33 MBytes on PE-0), but the sum of all intermediate result sizes is around 5 Gbytes and is greater than the final result size. This is not a huge amount of data, but future applications aiming to improve EDF electricity asset management, will manage greater data than this first benchmark application! Moreover, when each process stores data in 3 or 5 files at each time step, thousands files can be simultaneously opened on the parallel computer. Depending on the file system configuration, some problems can happen. On our Blue Gene/L system it has been mandatory to schedule the IO operations (to avoid to tune the file system just for our application). So, the storage of our intermediate results has succeeded, but remains a serious topic in our application.

The *final* results are stored in 3 files on one disk by process 0, and have a total size of 807.2 MBytes. In the current version of our application, the final result values are computed at each step of the *simulation* part by all processes, and gathered and stored by process 0 at each step. We use this mechanism because it does not require any specific file system, and is highly portable.

During the *optimization* part of the application, our load balancing strategy is based on a redistribution of the *N-cube* of data and associated computations at each time step. These successive distributions are achieved by the *N-cube splitting algorithm* introduced in section 3.2.3. A fixed number of MPI processes are started at the beginning of the application run, and our splitting algorithm attempts to provision a maximum of these MPI processes with *cubic* subcubes of the initial *N-cube*. However, the number of visited stock points changes due to the appearance and disappearance of some managed futures stocks, and it is not always possible to provision all MPI processes at each time step.

Figure 12 shows the number of visited stock points (corresponding to the size of the problem), and the number of provisioned processes on our Blue Gene/L and our PC cluster testbeds. The most efficient implementation on the PC cluster consists in running 1 MPI process per node and 2 threads per node (this cluster has 256 nodes and 2 cores per node). The provisioning is excellent: we run only 256 MPI processes and they are all provisioned at each time step. Our Blue Gene/L supercomputer has 4096 dual-core nodes and does not support multithreading, so we have to run 8192 MPI processes to use all cores. We can observe on figure 12 the MPI processes are not all provisioned at each step. There are many time steps with only 2048 MPI processes provisioned. In this experiment these steps correspond to a limited number of visited stock points. So, they are quickly processed and do not impact significantly the performances of the execution. This provisioning curve illustrates both the ability of our load balancing mechanism to dynamically adapt to important changes, and the limit of our distribution strategy on very large architectures. It can be noted our application easily supplies its provisioning performances in order we can study and take care about this important issue, especially when running on a large scale Blue Gene architecture.

4 Conclusion and perspectives

Current results: From a parallel computing point of view, we have designed and validated a distributed algorithm of stochastic control and simulation for $N-$dimensional problems. This algorithm includes complex and frequent communications and file IO operations, and a complete redistribution of data and computations at each time step. In order to achieve best performances, our algorithm use asynchronous communications and explicit synchronization mechanisms to overlap all communications and to use the interconnection networks of distributed machines at their maximal capacities.

This strategy has achieved good speedup and size-up on PC clusters and Blue Gene architectures, allowing to run a 7-stock and 10-state-variable benchmark in less than 2 hours. This benchmark requires too much memory to run on a classic sequential machine, and had never been run before. Moreover, the different performance curves we have obtained show a regular decrease of the execution time function of the number of nodes and cores. Finally, our distributed application exhibits a good scalability.

Our stochastic control and simulation algorithm has been implemented using a combination of two parallel programming paradigms: message passing (with MPI) was used to distribute computations and data on a large number of computing nodes while multithreading (with OpenMP or Intel TBB) was used to parallelize the node computations on several cores. A unique source code is available for both multicore-PC clusters and IBM Blue Gene architectures (BG/L and BG/P), making easier the code maintenance.

From a user point of view, our algorithm and its implementation support different price models for the gaz storage valuation case. It has been proved by this study that the curse of dimensionality can be postponed while using the Stochastic Dynamic Programming method in optimization. Moreover, we have applied our algorithm to an asset management problem for a power utility that has to maximize its profits by managing its power thermal assets, some stocks of water and some future contracts on the market. This approach is useful for research. It gives us reference calculation that has already permitted us to derive some efficient heuristics to treat our problem with a cost nearly linear with the number of futures used [23].

Next steps and objectives: In a close future, we will try to optimize the source code on Blue Gene architecture with the help of IBM, in order to improve the absolute performances, and we will run more benchmarks on large scale distributed architectures such as the entire 32,784-core Blue Gene/P of EDF company or some clusters of clusters on Grid'5000 (the French experimental grid). All these experiments will help us to point out the limits of our current algorithm and implementation, both in term of speedup and size-up.

But, from an application viewpoint our main remains to test our application on real data from the producer which corresponds to one hundred thermal assets with four stocks of water. We will test some modelings of the uncertainties in order to see their effects on simulation. Then we will increase the number of water stocks and add some futures contracts with the heuristic tested in [23]. It will allow us to treat more water stocks and it will permit us to describe more accurately our portfolio, generating more gains on average and saving some energy stocks.

Acknowledgment

This research is part of the ANR-CIGC GCPMF project, and is supported both by ANR (French National Research Agency) and by Region Lorraine.

References

[1] Boost C++ libraries, Getting Started. http://www.boost.org/doc/, 2008.

[2] Intel(R) threading building blocks, reference manual. Technical Report 315415-001US, INTEL, 2008.

[3] O. E. Barndorff-Nielsen. Processes of Normal Inverse Gaussian Type. *Finance and stochastics*, 2, 1998.

[4] R. E. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, 1957.

[5] T. Christiansen. Financial risk management in the electric power industry using stochastic optimization. *AMO - Advanced Modeling and Optimization*, 16(2), 2004.

[6] J. Hull. *Options, Futures, and other derivative.* Prentice Hall, 2005.

[7] P. Jaillet, E. Ronn, and S. Tompaidis. Valuation of commodity-based swing options. *Management science*, 50, 2004.

[8] G. Cohen J.C. Culioli. Decomposition-coordination algorithms in stochastic optimization. *SIAM Journal of Control and Optimization*, 28(6), 1990.

[9] T. Sargent L. Ljungqvist. *Recursive Macroeconomic Theory.* MIT Press, 2004.

[10] F. Longstaff and E. Schwartz. Valuing american options by simulation : A simple least-squares. *Review of Financial Studies*, 14(1), Spring 2001.

[11] C. Makassikis, S. Vialle, and X. Warin. Large scale distribution of stochastic control algorithms for financial applications. In *The First International Workshop on Parallel and Distributed Computing in Finance*, Miami, USA, April 2008.

[12] C. Makassikis, X. Warin, and S. Vialle. Distribution of a stochastic control algorithm applied to gas storage valuation. In *The 7th IEEE International Symposium on Signal Processing and Information Technology*, Cairo, Egypt, 2007.

[13] M. Mascagni and A. Srinivasan. Algorithm 806: SPRNG: a scalable library for pseudorandom number generation. *ACM Trans. Math. Softw.*, 26(3), 2000.

[14] R. Munos. Performance bounds in lp norm for approximate value iteration. *SIAM Journal of Control and Optimization*, 46(2), 2008.

[15] E. Prescott N. Stokey, R. E. Lucas. *Recursive Methods in Economic Dynamics.* Harvard University Press, 1989.

[16] P.S. Pacheco. *Parallel programming with MPI.* Morgan Kaufmann, 1997.

[17] D. Kohr D. Maydan J. McDonald R. Chandra, L. Dagum and R. Menon. *Parallel PRogramming in OpenMP.* Morgan Kaufmann, 2001.

[18] X. Warin S. Vialle and P. Mercier. A n-dimensional stochastic control algorithm for electricity asset management on pc cluster and blue gene supercomputer. In *9th International Workshop on State-of-the-Art in Scientific and Parallel Computing (PARA08)*, NTNU, Trondheim, Norway, May 13-16 2008.

[19] A. Gjelsvik T. A. Rotting. Stochastic dual dynamic programming for seasonal scheduling in the norwegian power system. *Transactions on power system*, 7(1), 1992.

[20] J.N. Tsitsiklis and B. Van Roy. Optimal stopping of markov processes: Hilbert spaces theory, approximations algorithms and an application to pricing high-dimensional financial derivatives. *IEEE Transactions on Automatic Control*, 44(10), Oct. 1999.

[21] T. Veldhuizen. Blitz++ User's Guide, Version 1.2. http://www.oonumerics.org/blitz/manual/blitz.html, February 2001.

[22] X. Warin. Gas storage modelling. Technical Report H-R33-2007-0-FR, EDF, 2007.

[23] X. Warin and W. van Ackooij. Electricity asset management with future hedging. Technical Report H-R33-2006-04103-FR, EDF, 2008.