Parallel solution of American option derivatives on GPU clusters

Lilia Khodja, Ming Chau, Raphaël Couturier, Jacques Bahi, Pierre Spitéri

Parallel solution of American option derivatives on GPU clusters

Lilia Ziane Khodja[a], Ming Chau[b], Raphaël Couturier[a], Jacques Bahi[a], Pierre Spiteri[c,*]

[a] *FEMTO-ST Institute, University of Franche Comte, IUT Belfort-Montbéliard, Rue Engel Gros, BP 527, 90016 Belfort CEDEX, France*
[b] *Advanced Solutions Accelerator, 199 rue de l'Oppidum, 34170 Castelnau Le Lez, France*
[c] *ENSEEIHT-IRIT, 2 rue Charles Camichel, 31071 Toulouse CEDEX, France*

Abstract

This paper deals with the numerical solution of financial applications, more specifically the computation of American option derivatives modelled by nonlinear boundary values problems. In such applications we have to solve large-scale algebraic systems. We concentrate on synchronous and asynchronous parallel iterative algorithms carried out on CPU and GPU networks. The properties of the operators arising in the discretized problem ensure the convergence of the parallel iterative synchronous and asynchronous algorithms. Computational experiments performed on CPU and GPU networks are presented and analyzed.

Keywords: Parallel asynchronous algorithms, iterative parallel numerical methods, subdomain method, sparse nonlinear systems, large scale obstacle problems, finance, GPU clusters, CUDA

1. Introduction

For the past few years financial applications have undergone great development. In particular, European and American options derivatives mod-

*Corresponding author
Email addresses: lilia.ziane_khoja@univ-fcomte.fr (Lilia Ziane Khodja), mchau@asa-sas.com (Ming Chau), raphael.couturier@univ-fcomte.fr (Raphaël Couturier), jacques.bahi@univ-fcomte.fr (Jacques Bahi), pierre.spiteri@enseeiht.fr (Pierre Spiteri)

elled by the classical evolutive Black and Scholes equation have been received with a great interest [28]. One should remember that the European option derivative is modelled by a classical time dependent convection-diffusion boundary value problem whereas the American option derivative is modelled by an evolutive variational inequality defined from a time dependent convection-diffusion operator. In what follows, since the solution of a parabolic convection-diffusion boundary value problem is very classical, we will not study the solution of European option derivative. We will rather focus on the determination of American option derivative. The problem consists also in solving a time dependent boundary value problem defined on an unbounded domain generally included in the three dimensional space. A classical artifice consists in solving the Black and Scholes equation on a bounded domain and in increasing the size of the domain to ensure the convergence to the exact solution [11]. Moreover the solution of such model problems belongs to a convex closed set, and consequently the determination of the solution needs a projection on this convex set. So, such boundary value problems need to be solved numerically.

The temporal discretization leads to solving a sequence of stationary nonlinear boundary value problems; after spatial discretization we have then to solve large-scale algebraic systems. Taking into account the size of the algebraic systems derived from the discretization process of the Black and Scholes equation, parallel iterative algorithms are well adapted. Therefore in this paper, we concentrate on asynchronous and synchronous iterative methods for solving these algebraic systems. So we consider two kinds of such parallel algorithms: on the one hand, the parallel projected Richardson algorithm associated to a problem of optimization and, on the other hand, the parallel block relaxation algorithm corresponding in fact to a subdomain method without overlapping.

The obstacle problem has been studied in many contributions and by many authors. The obstacle problem is more general than the problem of American option derivative. It appears in many applications such as mechanics, free boundary value problem and finance. For example, in [15] sequential methods are considered for the Hamilton-Jacobi-Bellman problem, this latter problem being related to the obstacle problem; in [26] a study concerning the rate of convergence when multilevel domain decomposition and multigrid methods is considered for sequential solution of the obstacle problem. In [1] the linear convergence is proved for a sequential multiplicative Schwarz method, applied to the solution of variational inequalities and in [2]

a geometric convergence rate is established for sequential additive Schwarz method for the same problem; in [24] the convergence rate analysis of domain decomposition methods is also studied in the case of obstacle problem solution. In [13] the block relaxation methods for algebraic obstacle problems with M-matrices are considered while in [12] multiplicative and additive Schwarz methods for obstacle problems with convection-diffusion operators, when two (possibly overlapping) subdomains are considered, one, where the solution equals to a given obstacle function, and the other, where the solution satisfies a linear equation. In [14] the sequential convergence analysis of the generalized Schwarz method for solving the obstacle problems with T-monotone operator are studied. In the previous contributions, note that, mainly, the study of numerical solutions is considered by using sequential subdomain methods. Furthermore, in the previous studies, when the parallel solution is considered for the solution of the obstacle problem, it can be noticed that only parallel synchronous subdomain methods is studied. In [25] the rate of convergence of parallel asynchronous methods is studied in the context of convex optimization; the considered work include as particular cases parallel domain decomposition and multigrid methods for solving elliptic partial differential equations and is closely related to relaxation methods for non linear network flow.

In the present study, note that we consider only a subdomain method with no overlapping between the subdomains. Moreover we consider also the implementation of such algorithms on various architectures like CPU and GPU architectures and we compare the performances of the studied computational methods on each architecture. It is one of the main contribution of the presented study. Indeed, since the objective is to solve quickly large-scale algebraic systems, the implementations of both iterative methods are better suited on parallel architectures. Nowadays, the most attractive parallel computers are those using the computing power of the GPU cards (Graphics Processing Units). Their hardware and software architectures have rapidly evolved, allowing them to become high performance accelerators for the data-parallel tasks and intensive arithmetic computations of many applications. Several works have proved the ability of the GPUs to provide better performances than the CPUs for many applications using iterative solvers [29], [16] and [8]. So, in the following we will compare the performances of the projected Richardson algorithm and the block relaxation algorithm, both implemented, on these two kinds of architectures: CPU and GPU clusters. From a practical point of view, note that the interest of the present study is to

provide financial analysts with fast and non-expensive computation facilities in their work.

The paper is organized as follows: section 2 is devoted to the presentation of the problem of American options derivatives; in particular using classical mathematical background in applied analysis, then we present equivalent formulations of the model problem. Section 3 presents the numerical solutions of the model problem, in particular appropriate discretization ensuring the convergence of parallel iterative algorithms and the general models of asynchronous methods; the parallel subdomain method without overlapping and the parallel projected Richardson algorithm are presented and analyzed in details. Finally, the parallel experiments on CPU and GPU architectures are presented with both synchronous and asynchronous versions.

In the sequel, the dot notation "." always represents products between two scalars, a scalar and a vector or a matrix and a vector. The inner product is denoted with brackets: $<,>$.

2. The problem of American options derivative

2.1. The problem to solve

We consider the case of American options modeled by the Black-Scholes equations [28]. The classical Black-Scholes equation is a boundary value problem describing the evolution of call or put options in the field of mathematics of financial contracts. Let us note that American options may be exercised at any time prior to expiration, i.e. when the time τ takes any value between 0 and T, where T denotes the expiry date. The Black-Scholes equation is classically written like the following retrograde time dependent nonlinear convection-diffusion equation, called also obstacle problem:

$$\begin{cases} \frac{\partial \bar{v}}{\partial \tau} + (r - \frac{\sigma^2}{2})\nabla \bar{v} + \frac{\sigma^2}{2}\Delta \bar{v} - r\bar{v} \geq 0,\ \bar{v} \geq \phi, e.w.\ in\ [0,T] \times \mathbb{R}^n \\ (\frac{\partial \bar{v}}{\partial \tau} + (r - \frac{\sigma^2}{2})\nabla \bar{v} + \frac{\sigma^2}{2}\Delta \bar{v} - r\bar{v})(\bar{v} - \phi) = 0, e.w.\ in\ [0,T] \times \mathbb{R}^n \\ \bar{v}(T,S) = \phi \end{cases} \quad (1)$$

where, e.w. means every where, $\phi = \phi(S) = \max(S - K, 0)$ in the case of call option or $\phi = \max(K - S, 0)$ in the case of put option; in the previous equations $\bar{v}$ denotes the value of the considered option at time τ, i.e. a call or a put option; $\bar{v} = \bar{v}(\tau, S)$ is a function of the current value of the underlying asset S and of the time τ. Note also that the considered problem (1) also depends on the following parameters:

- r the interest rate,
- σ the volatility of the underlying asset, σ being in fact the instantaneous standard deviation of the price with respect to the exercise price K, classically called strike and fixed beforehand; in fact σ characterizes the uncertainty of the option's behavior. Δ denotes the Laplacian operator and ∇ denotes the gradient operator.

Note that the previous boundary value problem is not defined on a bounded domain, but is defined on the unbounded domain $\mathbb{R}^n, n \geq 1$. This difficulty is solved by considering the problem defined on a bounded domain $\Omega \subset \mathbb{R}^n$ and it can be proven that the solution of the retrograde time dependent convection-diffusion equation, defined on the bounded domain Ω, converges to the solution of problem (1) when the measure of Ω tends to infinity (see [11]). This previous issue can be solved, by choosing Ω like $\Omega = [-L, L]^n$, where L is a parameter which can be as large as possible according to the use of the result of [11].

Another particularity of the problem to solve is that the value of the option is not known at the initial time $\tau = 0$; only the value $\bar{v}(T, S)$ is known and the problem consists in computing $\bar{v}(0, S)$. Obviously, this feature can be solved easily, by considering a change of variables concerning the time and replacing the variable τ by a variable $t = T - \tau$.

Thus, problem (1) is then replaced by a classical time dependent convection-diffusion problem modeled as follows:

$$\begin{cases} \frac{\partial \bar{v}}{\partial t} - (r - \frac{\sigma^2}{2})\nabla \bar{v} - \frac{\sigma^2}{2}\Delta \bar{v} + r\bar{v} \geq 0,\ \bar{v} \geq \phi, e.w.\ in\ [0,T] \times \Omega \\ (\frac{\partial \bar{v}}{\partial t} - (r - \frac{\sigma^2}{2})\nabla \bar{v} - \frac{\sigma^2}{2}\Delta \bar{v} + r\bar{v})(\bar{v} - \phi) = 0, e.w.\ in\ [0,T] \times \Omega \\ \bar{v}(0, S) = \phi \\ B.C.\ on\ \bar{v}(t, S) \text{ defined on } \partial\Omega \end{cases} \tag{2}$$

where B.C. describes the boundary conditions on the boundary $\partial\Omega$ of the domain Ω. Practically, the Dirichlet condition (where $\bar{v}$ is fixed on $\partial\Omega$) or the Neumann condition (where the normal derivative of $\bar{v}$ is fixed on $\partial\Omega$) are classically considered.

Thus, using an implicit or semi-implicit time marching scheme, the problem consists in solving numerically a sequence of stationary non linear convection-diffusion problems written as follows:

$$\begin{cases} -(r - \frac{\sigma^2}{2})\nabla v - \frac{\sigma^2}{2}\Delta v + (r + c)v - g \geq 0,\ v \geq \phi, e.w.\ in\ \Omega, \\ (-(r - \frac{\sigma^2}{2})\nabla v - \frac{\sigma^2}{2}\Delta v + (r + c)v - g)(v - \phi) = 0, e.w.\ in\ \Omega, \\ B.C.\ \text{on } v(t, S) \text{ defined on } \partial\Omega, \end{cases} \tag{3}$$

where $v = v(S)$ denotes the value of $\bar{v}(t, S)$ at current time t, c is the inverse of the time step k and $g = \frac{1}{k} \cdot v^{prec}$, where v^{prec} is the solution obtained at the previous time step.

Remark 1. *Note that the convection-diffusion operator is not self-adjoint. Nevertheless, in what follows, for the numerical solution of the discrete obstacle problem, the fact that the operator, arising in the obstacle problem, is self-adjoint or is not self-adjoint, plays a main role in the choice of the appropriate algorithm needed for the numerical solution of the resulting algebraic systems to solve. So, since the coefficients arising in the operator are constant, then by a classical change of variables, we can replace the stationary convection-diffusion operator*

$$b^t \nabla v - \nu \Delta v + (r + c)v = g, e.w. \text{ in } \Omega, \ c = \frac{1}{k} > 0,$$

where b is a constant coefficient of convection, ν is a constant coefficient of diffusion and c is a positive coefficient, by a diffusion operator; indeed, the change of variables

$$v = e^a \cdot u,$$

where a is defined by $a = \frac{b^t S}{2\nu}$, leads to the following expression of this operator

$$-\nu \Delta u + (\frac{\|b\|_2^2}{4\nu} + c)u = e^{-a}.g = f,$$

corresponding to a self adjoint operator. In this case, note that $u \in K$ is classically a solution of the following optimization problem

$$\begin{cases} Find\ u \in K such\ that \\ J(u) \leq J(w), \forall w \in K \end{cases} \tag{4}$$

where

$$J(w) = \frac{1}{2} a(w, w) - L(w),$$

where

$$a(u, w) = \int_\Omega (\frac{\nu}{2}.\nabla u.\nabla w + (\frac{\|b\|_2^2}{4\nu} + c).u.w) dxdydz, c \geq 0,$$

and

$$L(w) = (f, w) = \int_\Omega f.w dxdydz,$$

and where K is a closed convex set which will be precised in what follows; lastly the two problems are equivalent and then any solution of the problem of optimization is also a solution of the diffusion problem and conversely.

2.2. Mathematical background

The notion of sub-differential mapping will play a major role in the following parts for taking into account the necessary projection on a closed convex set K. So, we recall hereafter this notion and its main associated properties (see [4]).

Definition 1. *Given a convex function χ on a normed vectorial space E and a point $u \in E$, we denote by $\partial\chi(u)$ the set of all $u' \in E'$ such that*

$$\chi(v) \geq \chi(u) + < v - u, u' >_{E \times E'} \text{ , for every } v \in E, \tag{5}$$

where $<$, $>$ denotes the pairing between E and E'. Such element u' is called subgradient of χ at u and $\partial\chi(u)$ is called the subdifferential of χ at u.

Remark 2. *Recall that the pairing between E and E' is a bilinear form, from $E \times E'$ onto $\mathbb{R}$ (or possibly $\mathbb{C}$). If E is an Hilbert space, then the pairing is the inner product of E.*

Remark 3. *Let χ be a Gateaux differentiable (or Frechet differentiable) convex mapping at u. Then $\partial\chi(u)$ consists of a single element, namely the Gateaux (or Frechet) differential of χ at u (see [4]). From (5), it is obvious that $\partial\chi(u)$ is a closed convex set (possibly empty, see [4]).*

In what follows, we will use a possibly multivalued formulation of the model problem of American options derivatives which play a major role for the numerical solution of the studied nonlinear problems when relaxation methods are used. Recall the following results:

Lemma 1. *Let $u \in E$; u is such that $\chi(u) = \min_{v \in E}(\chi(v))$ if and only if $0 \in \partial\chi(u)$. Moreover, the subdifferential $\partial\chi(u)$ is a monotone operator (in general multivalued) from E to E'.*

Proof. Indeed, let $u \in E$ such that $\chi(u) \leq \chi(v), \forall v \in E$; then we have $\chi(v) \geq \chi(u) + < v - u, 0 >_{E \times E'}$, and then $0 \in \partial\chi(u)$. Besides, let $w' \in \partial\chi(w)$; then $\chi(v) \geq \chi(w) + < v - w, w' >_{E \times E'}, \forall v \in E$. Let also $u' \in \partial\chi(u)$; then $\chi(v) \geq \chi(u) + < v - u, u' >_{E \times E'}, \forall y \in E$. Consider the first inequality for $v = u$ and the second for $v = w$; then by adding, we obtain $< w - u, w' - u' >_{E \times E'} \geq 0$, and the proof is achieved. ■

The indicator function of the convex set K will also be used in the sequel. The indicator function of the convex subset K is defined as follows

Definition 2. *Let K be a closed convex subset of E and let ψ_K be the indicator function of the convex subset K, i.e.*

$$\psi_K(u) = \begin{cases} 0 \text{ if } u \in K \\ +\infty \text{ otherwise} \end{cases}$$

Clearly, $\psi_K(u)$ is convex. By definition of the subdifferential we have (see [4])

$$\partial\psi_K(v) = \{v' \in E' | < v - w, v' >_{E\times E'} \geq 0, \text{ for every } w \in K\}.$$

This shows that $D(\partial\psi_K) = D(\psi_K) = K$ and $\partial\psi_K(v) = \{0\}$ for each $v \in int(K)$, where $int(K)$ denotes the interior of the set K. Moreover, if v lies on the boundary of K, then $\partial\psi_K(v)$ coincides with the cone normal to K at point v.

2.3. Equivalent formulations of the model problem

For each stationary obstacle problem (3) many equivalent formulations of the obstacle problem exist and the reader is referred to [10] for complements. In the present subsection, we concentrate now on the variational formulation associated with problem (3); classically the corresponding variational formulation is given as follows

$$\begin{cases} \text{Find } v \in K \text{such that} \\ a(v, w - v) \geq L(w - v), \forall w \in K \end{cases} \tag{6}$$

where, with the same notations as the ones used in remark 1:

$$a(v, w) = \int_\Omega (\frac{\sigma^2}{2}.\nabla v.\nabla w - (r - \frac{\sigma^2}{2}).\nabla v.w + (r + c).v.w)dxdydz,$$

$$L(w) = (f, w) = \int_\Omega f.wdxdydz,$$

and

$$K = \{w | w \text{ given in } \Omega, \text{ such that } w(x, y, z) \geq \phi(x, y, z) \text{ everywhere on } \Omega\}.$$

Consider the stationary variational inequality (6). Then, since classically

1. the space $\mathcal{H}^1(\Omega)$ normed by $\|w\|^2_{1,\Omega} = \int_\Omega (\nabla w.\nabla w + w.w)dxdydz$ is classically an Hilbert space and K is obviously a closed convex set,

2. the mapping $(v, w) \rightarrow a(v, w)$ is obviously a bilinear, continuous and elliptic form,
3. the mapping $w \rightarrow L(w)$ is obviously a linear continuous form,

then, by applying the classical Stampacchia theorem, the variational inequality (6) has a unique solution.

We can also apply the Riesz representative theorem to the formulation of the problem (6). Let us denote by $E = \mathcal{H}^1(\Omega)$ the real vectorial space and by E' its dual space; in our case E' is identified with E, since $\mathcal{H}^1(\Omega)$ is an Hilbert space. Then, there exists a unique element denoted $\bar{A}v \in E'$, where $\bar{A}$ is a continuous linear mapping from $E \rightarrow E'$, such that

$$a(v, w) =< w, \bar{A}v >_{E \times E'}, \forall w \in \mathcal{H}^1(\Omega);$$

similarly, there exists also an unique element, denoted $\bar{g} \in E'$ such that

$$L(w) =< w, \bar{g} >_{E \times E'}, \forall w \in \mathcal{H}^1(\Omega);$$

then the stationary variational inequality (6) can be written as follows

$$\left\{ \begin{array}{l} \text{Find } v \in K \text{such that} \\ < w - v, \bar{A}v - \bar{g} >_{E \times E'} \geq 0, \forall w \in K. \end{array} \right. \tag{7}$$

Remark 4. *From a practical point of view, note that for the studied stationary problem (6)* $\bar{A}v(x, y, z) = -\frac{\sigma^2}{2}.\Delta v(x, y, z) - (r - \frac{\sigma^2}{2}).\nabla v(x, y, z) + (r + c).v(x, y, z)$ *and* $\bar{g} = g(x, y, z)$.

Then the new formulation (7) of the stationary variational inequality (6) is equivalent to the following multivalued problem

$$\left\{ \begin{array}{l} \text{Find } v \in E \text{ such that} \\ \bar{A}v - \bar{g} + \partial\psi_K(v) \ni 0, \end{array} \right. \tag{8}$$

where $\partial\psi_K(v)$ is the subdifferential of the indicator function of the convex subset K. Note that in (8), the projection on the convex set K is formulated by the perturbation of the continuous convection-diffusion operator by a multivalued increasing (or monotone) diagonal operator. Indeed, the definition of the subdifferential of the indicator function implies

$$\partial\psi_K(v) = \{\omega \in E' | \psi_K(w) - \psi_K(v) \geq < w - v, \omega >_{E \times E'}, \forall w \in K\};$$

assume that (7) holds and let us verify that the following inequality is true

$$\psi_K(w) - \psi_K(v) \geq < w - v, \bar{g} - \bar{A}v >_{E\times E'} , \forall w \in K. \tag{9}$$

If $w \in K$, then $\psi_K(w) = 0$ and since $\psi_K(v) = 0$, then we obtain

$$0 \geq < w - v, \bar{g} - \bar{A}v >_{E\times E'},$$

inequality valid by considering (7). If $w \notin K$, then $\psi_K(w) = +\infty$; since $\psi_K(v) = 0$, then we obtain

$$+\infty - 0 \geq < w - v, \bar{g} - \bar{A}v >_{E\times E'},$$

and the obtained inequality is always verified.

Conversely, if there exists $v \in E$ such that $\bar{g} - \bar{A}v \in \partial\psi_K(v)$ then we have to verify that there exists $v \in E$ such that (9) holds for all $w \in K$. If $v \in K$ then $\psi_K(v) = 0$ and (9) can be written as follows

$$\psi_K(w) \geq < w - v, \bar{g} - \bar{A}v >_{E\times E'} , \forall w \in K.$$

Then since $w \in K$ then $\psi_K(w) = 0$ and (9) can be written as follows

$$0 \geq < w - v, \bar{g} - \bar{A}v >_{E\times E'} , \forall w \in K,$$

inequality well verified by (7).
Besides, let us show that the assertion $v \notin K$ is impossible. Indeed, in this case $\psi_K(v) = +\infty$, so that

$$\psi_K(w) - \infty \geq < w - v, \bar{g} - \bar{A}v >_{E\times E'} , \forall w \in K,$$

which involves, since $\psi_K(w) = 0$, that

$$-\infty \geq < w - v, \bar{g} - \bar{A}v >_{E\times E'} , \forall w \in K$$

inequality never verified. In conclusion, the problems (8) and (9) are equivalent. So the numerical solution of the obstacle problem leads to solving the multivalued problem (8) and conversely.

Remark 5. *If in the formulation of problem (6), the term of convection is zero, the previous multivalued formulation still holds, in which in conformity with Remark 1, the associated operator is a diffusion operator, then of the following form* $\bar{A}u(x, y, z) = -\frac{\sigma^2}{2}.\Delta u(x, y, z) + (\frac{\|b\|_2^2}{4\nu} + c).u(x, y, z)$.

3. Numerical solution

For the numerical solution of the problem of American option pricing, we will consider the use of two particular parallel iterative fixed point methods: the parallel projected subdomain relaxation method without overlapping associated to the convection-diffusion problem or the parallel projected Richardson method associated only to the diffusion problem, this last problem being related to the solution of an optimization problem in conformity with Remark 1. Thanks to appropriate discretization of the spatial part of these two previous problems, the discrete operator has a common property ensuring the convergence of the two parallel iterative methods considered.

3.1. Discretization of the operators

After appropriate temporal discretization by an implicit time marching scheme, the problem consists in solving numerically a sequence of stationary non linear convection-diffusion problems in the case where we do not use the change of variables presented in subsection 2.1 or a sequence of stationary non linear diffusion problems otherwise.

In the parallel experiments, both problems are solved in a constant domain Ω in order to test the studied parallel methods. The parameters of the convection-diffusion operator are: $\frac{\sigma^2}{2} = 0.2$ and $r = 1.1$. In addition, 3 time steps are computed with $k = 0.0066$. The initial function $\bar{v}(0, x, y, z)$ of obstacle problem (2) is 0, with a constraint $\bar{v} \geq \phi = 0$. In order to obtain significant data for performance analysis, so that the numbers of relaxations required to solve the non linear systems at every time step are sufficiently high, an additional source term

$$g'(x, y, z) = \cos(2\pi x).\cos(4\pi y).\cos(6\pi z)$$

is added to the right hand side of the convection-diffusion equation. Then equation (3) is slightly modified by:

$$g = \frac{1}{k} \cdot v^{prec} + g'. \tag{10}$$

Note that, the initial guess for iterative algorithms is set to the solution found at the previous time step. Without this additional source term, $\bar{v}(0, x, y, z)$ should be nonzero, but convergence would occur too rapidly.

In the following, we will consider that $\Omega \subset \mathbb{R}^3$ is discretized with an uniform Cartesian mesh constituted by $M = m^3$ discretization points, where m

is related to the spatial discretization step h, so that the complete discretization of the boundary value problem (2) leads to the solution of the following large stationary discrete variational inequality at each time step

$$\begin{cases} \text{Find } V^* \in K \text{ such that} \\ \forall W \in K, \langle \mathcal{A}.V^*, W - V^* \rangle \geq \langle G, W - V^* \rangle, \end{cases} \tag{11}$$

where $\mathcal{A} = A + \frac{1}{k}I$, A is the block matrix obtained after spatial discretization by finite difference method, k is the time step, K is the discrete convex set defined by

$$K = \{V \mid V \geq \bar{\Phi} \text{ everywhere in } E\}$$

in which $\bar{\Phi}$ is the discretized obstacle function, G is derived from the Euler first order implicit time marching scheme, from the discretized right-hand side of the obstacle problem and I is the identity matrix. More precisely, note also that the spatial differential operators are discretized as follows:
- the Laplacian is discretized by using the classical seven points scheme when $\Omega \subset \mathbb{R}^3$,
- when the first derivatives are considered in the model problem (i.e. the convection phenomenon is taken into account) they are discretized by using decentered scheme; in fact, if the coefficient of the considered first derivative is strictly positive, then the forward-difference formula is used, else backward difference formula is considered.
Thus, in both cases A is an M-matrix (see [27]) and consequently the matrix $(A + \frac{1}{k}I)$ is also an M-matrix.

Note that we have to solve a variational inequality indifferently when the Dirichlet boundary condition or the Neumann boundary condition are considered; nevertheless, in the sequel, we consider that the boundary conditions are the Dirichlet ones, but, when appropriate assumptions are satisfied, the present study is still valid if we consider the Neumann boundary conditions. Moreover, since the global discretized matrix is an M-matrix, then we will show in the following that this property ensures the convergence of the studied parallel asynchronous or synchronous algorithms (see [18] to [20]).

3.2. Parallel iterative methods

In the following part, we will present the solution, at each new time step, of the next discrete complementary problem (26) by various parallel synchronous or asynchronous iterative algorithms(see [3, 5, 6, 7, 17]).

Let α be a positive integer. Assume that $E = \mathbb{R}^M$; note that E is an Hilbert space. Consider also that the space $E = \prod_{i=1}^{\alpha} E_i$ is a product of α subspaces denoted $E_i = \mathbb{R}^{m_i}$, where $\sum_{i=1}^{\alpha} m_i = M$; note that E_i is also an Hilbert space in which $\langle \, . \, , \, . \, \rangle_i$ denotes the scalar product and $| \, . \, |_i$ the associated norm, for all $i \in \{1, \ldots, \alpha\}$.

Let $W \in E$ and consider the following block decomposition of W (see figure 6) and the corresponding decomposition of F

$$\begin{array}{rcl} W & = & (W_1, \ldots, W_\alpha) \\ F(W) & = & (F_1(W), \ldots, F_\alpha(W)) \end{array}$$

Then for all $U, W \in E$ let us denote by $\langle U \, , W \rangle = \sum_{i=1}^{\alpha} \langle U_i \, , W_i \rangle_i$ the scalar product on E and $\| \, . \, \|$ its associated norm.

In the sequel, we consider the general following fixed point problem

$$\left\{ \begin{array}{l} \text{Find } W^* \in E \text{ such that} \\ W^* = F(W^*) \end{array} \right. \tag{12}$$

where $W \mapsto F(W)$ applies from E to E.

In order to solve the fixed point problem (12), let us consider now the parallel asynchronous iterations defined as follows: let $W^0 \in E$ be given, then for all $p \in \mathbb{N}$, W^{p+1} is recursively defined by

$$W_i^{p+1} = \left\{ \begin{array}{l} F_i(W_1^{\rho_1(p)}, \ldots, W_j^{\rho_j(p)}, \ldots, W_\alpha^{\rho_\alpha(p)}) \text{ if } i \in s(p) \\ W_i^p \text{ if } i \notin s(p) \end{array} \right. \tag{13}$$

where

$$\left\{ \begin{array}{l} \forall p \in \mathbb{N}, s(p) \subset \{1, \ldots, \alpha\} \text{ and } s(p) \neq \emptyset \\ \forall i \in \{1, \ldots, \alpha\}, \{p \mid i \in s(p)\} \text{ is countable} \end{array} \right. \tag{14}$$

and $\forall j \in \{1, \ldots, \alpha\}$,

$$\left\{ \begin{array}{l} \forall p \in \mathbb{N}, \rho_j(p) \in \mathbb{N}, 0 \leq \rho_j(p) \leq p \text{ and } \rho_j(p) = p \text{ if } j \in s(p) \\ \lim\limits_{p \to \infty} \rho_j(p) = +\infty. \end{array} \right. \tag{15}$$

The previous asynchronous iterative scheme models computations that are carried out in parallel without order nor synchronization and describes

a subdomain method without overlapping. It enables particularly, to consider distributed computations whereby processors compute at their own pace according to their intrinsic characteristics and computational load. The parallelism between the processors is well described by the set $s(p)$ which contains at each step p the index of the components relaxed by each processor on a parallel way while the use of delayed components in (13) permits one to model nondeterministic behavior and does not imply inefficiency of the considered distributed scheme of computation. Note that, according to [17], theoretically, each component of the vector must be relaxed an infinity of time. The choice of the relaxed components may be guided by any criterion, and, in particular, a natural criterion is to pick-up the most recently available values of the components computed by the processors.

Remark 6. *Such asynchronous iterations describe various classes of parallel algorithms, such as parallel synchronous iterations if* $\forall j \in \{1, \ldots, \alpha\}$, $\forall p \in \mathbb{N}, \rho_j(p) = p$. *In the synchronous context, for particular choice of* $s(p)$*, then (13)-(15) describe classical sequential algorithms. Among them, the Jacobi method (*$\forall p \in \mathbb{N}, s(p) = \{1, \ldots, \alpha\}$*), and the Gauss-Seidel method (*$\forall p \in \mathbb{N}, s(p) = \{1 + p \bmod \alpha\}$*) (see [17]).*

Remark 7. *In order to measure the amount of computation required to reach convergence, we will use in the presented results of parallel experiments (see subsection 4.3) the number of relaxations instead of the number of iterations. A relaxation is the update (13) of a local iterate vector component according to* F_i*. This definition holds in both sequential, synchronous and asynchronous cases. An iteration is the update of at least all components with* F_i *in a parallel-synchronous or sequential way. Since this definition of iteration cannot be extended simply to the asynchronous case, we prefer using relaxation count as an indicator of the amount of floating operations required to reach convergence.*

3.3. Projected parallel synchronous and asynchronous relaxation method

For solving the model problem, we can first consider a projected parallel asynchronous block relaxation algorithm, related to the natural block decomposition of the discretized convection-diffusion operator; note also that this kind of method can be used for the solution of the discretized diffusion operator. This method corresponds in fact to a parallel subdomain relaxation method without overlapping.

In order to define the projected parallel asynchronous subdomain method, we complete the formalism introduced in subsection 3.2; let us denote by $W \equiv V$ the iterate vector and let us proceed as follows. Assume that $\forall i \in \{1, \ldots, \alpha\}, K_i \subset E_i$ where K_i is a closed convex set, and let $K = \prod_{i=1}^{\alpha} K_i$. Let also $G = (G_1, \ldots, G_\alpha) \in E$. For any $V \in E$, let $P_K(V)$ be the projection of V on K such that $P_K(V) = (P_{K_1}(V_1), \ldots, P_{K_\alpha}(V_\alpha))$ where $\forall i \in \{1, \ldots, \alpha\}, P_{K_i}$ is the projector from E_i onto K_i.

Since we consider a block decomposition of the problem to solve, so, let us denote by $\mathcal{A}_{i,i}$ the diagonal blocks of the matrix $\mathcal{A} = A + \frac{1}{k}I$, and by $\mathcal{A}_{i,j}$ $(j \neq i)$ the off-diagonal blocks; let us also denote by $\|\mathcal{A}_{i,j}\|_j$ the matrix norm of $\mathcal{A}_{i,j}$. Besides, thanks to the result presented in the previous subsection, the model problem is also formulated like the multivalued problem (8), which can be decomposed as follows

$$\mathcal{A}_{i,i}.V_i^\star - G_i + \sum_{j \neq i} \mathcal{A}_{i,j}.V_j^\star + \partial(\psi_K)_i(V_i^\star) \ni 0, \forall i \in \{1, \ldots, \alpha\}, \tag{16}$$

or

$$\mathcal{A}_{i,i}.V_i^\star - G_i + \sum_{j \neq i} \mathcal{A}_{i,j}.V_j^\star + \omega_i = 0, \omega_i \in \partial(\psi_K)_i(V_i^\star), \forall i \in \{1, \ldots, \alpha\}; \tag{17}$$

this last relation is associated to the following fixed point mapping

$$V_i^\star = P_{K_i}(\mathcal{A}_{i,i}^{-1}(G_i - \sum_{j \neq i} \mathcal{A}_{i,j} V_j^\star)) = F_{B_i}(V^\star), \forall i \in \{1, \ldots, \alpha\}. \tag{18}$$

Then we can associate to this fixed point mapping F_B a parallel asynchronous block method defined by (13)-(15).

Then, in the sequel we will assume the following assumptions

$$\forall V_i \in E_i, \langle \mathcal{A}_{i,i}.V_i \, , V_i \rangle \geq n_{i,i} |V_i|_i^2 \tag{19}$$

and

$$\|\mathcal{A}_{i,j}\|_j \leq -n_{i,j} \ , \ (n_{i,j} \leq 0), \tag{20}$$

Note that (19) corresponds to the fact that the diagonal blocks are positive definite and (20) means that the matrix norms of off-diagonal blocks are bounded, these two latter properties being always true, on the one hand, by

considering the previous appropriate spatial discretization schemes (see [9] and [23]) and, on the other hand, since the problem is formulated in finite dimension. Assume also that

$$N = (n_{i,j})_{1 \leq i,j \leq \alpha} \text{ is an } \alpha \times \alpha \text{ M-matrix.} \tag{21}$$

Then, we can state the following convergence result of the synchronous and asynchronous parallel block relaxation method

Proposition 1. *Assume that (19), (20) and (21) hold. Then the synchronous and the asynchronous iterations defined by (13), (14) and (15), applied with the fixed point mapping F_{B_i} defined by (18) and associated to the problem (8), converge to the limit V^* for every initial guess V^0, V^* being the unique solution of problem (26).*

Proof. Indeed, let us decompose the problem (8) into α sub problems; let us write the i^{th} block equation (17) verified, on the one hand, for the exact solution and, on the other hand, for a current value of the $(p+1)^{th}$ relaxed component; substracting and multiplying each of the α sub problems by $(V_i^\star - V_i^{p+1})$, leads to

$$\begin{aligned} &< \mathcal{A}_{i,i}.(V_i^\star - V_i^{p+1}), V_i^\star - V_i^{p+1} >_i + < \omega_i^\star - \omega_i^{p+1}, V_i^\star - V_i^{p+1} >_i \\ &\leq \sum_{j \neq i} | < \mathcal{A}_{i,j}.(V_j^\star - \tilde{V}_j), V_i^\star - V_i^{p+1} >_i, \forall i \in \{1, \ldots, \alpha\}, \end{aligned}$$

where $\omega_i^\star \in \partial(\psi_K)_i(V_i^\star)$ and $\omega_i^{p+1} \in \partial(\psi_K)_i(V_i^{p+1})$, and $\tilde{V}_j$ are the available current values produced by the other processors according to the algorithm (13)-(15); then, the left hand-side of the previous inequality can be minored by $n_{i,i}.|V_i^\star - V_i^{p+1}|_{2,i}^2$, since the subdifferential mapping is monotone, on the one hand, and using (19) on the other hand; while the right-hand side of the inequality can be majored using (20). Finally, we obtain

$$|V_i^\star - V_i^{p+1}|_{2,i} \leq \sum_{j \neq i} -\frac{n_{i,j}}{n_{i,i}} |V_j^\star - \tilde{V}_j|_{2,j}, \forall i \in \{1, \ldots, \alpha\}. \tag{22}$$

Note that the matrix J

$$J = \left\{ \begin{array}{rcl} J_{i,i} & = & 0 \\ J_{i,j} & = & -\frac{n_{i,j}}{n_{i,i}},\ i \neq j. \end{array} \right.$$

which diagonal entries are null and off-diagonal entries are equal to $-\frac{n_{i,j}}{n_{i,i}}$ is the Jacobi matrix of the matrix N. Since the matrix N is an M-matrix, then J is a non negative matrix with all modulus of the eigenvalues inferior to one. Let us denote by ν the spectral radius of J and by Θ the associated eigenvector. Classically by the Perron-Frobenius theorem, we have

$$J.\Theta = \nu\Theta \text{ and } \Theta > 0, \tag{23}$$

where all the components of Θ are strictly positive. Then, in a straightforward way, we obtain

$$\|V^\star - V^{p+1}\|_{\nu,\Theta} \leq \nu\|V^\star - \tilde{V}\|_{\nu,\Theta}, \forall \tilde{V} \in E, 0 \leq \nu < 1, \tag{24}$$

where $\|W\|_{\nu,\Theta}$, is a uniform weighted norm defined as follows

$$\|W\|_{\nu,\Theta} = \max_{i=1,\ldots,\alpha}(\frac{|W_i|_{2,i}}{\Theta_i}); \tag{25}$$

thus, (24) shows that F_B is a contraction and according to [17] the block parallel asynchronous, synchronous and sequential iterations described by (13)-(15) and associated to the fixed point mapping F_B, converge to $V^\star$ whatever the initial guess V^0 is, and the proof is achieved ■

Remark 8. *In the previous result, we have considered that all the spaces are normed by the Euclidean norm. In fact, we can also use the norms $l_k, 1 \leq k \leq \infty$ to prove the convergence of the block parallel asynchronous, synchronous and sequential iterations [23]. Nevertheless, in this case, the norms are nonhilbertian norms and the theoretical context changes. In this more general context, the scalar product is replaced by the usual bilinear form associated to a pair of dual spaces E_i and E_i', classically called the pairing between E_i and E_i'; so, $V_i^\star - V_i^{p+1}$ is replaced by $g_i(V_i^\star - V_i^{p+1}) \in \mathcal{G}_i(V_i^\star - V_i^{p+1})$, where $\mathcal{G}$ denotes the duality map of E and $\mathcal{G}_i(V_i^\star - V_i^{p+1})$ is accordingly the duality map of E_i; in fact, according to the Hahn-Banach theorem, it can be proved classically that $\mathcal{G}_i(V_i^\star - V_i^{p+1})$ is a closed nonempty set; furthermore the subdifferential mapping of the nonhilbertian norm in E_i, $\frac{1}{2}.|V_i^\star - V_i^{p+1}|_i^2$, coincides with $\mathcal{G}_i(V_i^\star - V_i^{p+1})$ (see [4]). The more practical norms to use are the l_1 norm and the l_∞ norm; in these two cases, inequality (19), in which $V_i^\star - V_i^{p+1}$ is replaced by $g_i(V_i^\star - V_i^{p+1})$, are still valid, but linked to properties of strictly diagonal dominance of the block diagonal submatrices $\mathcal{A}_{i,i}$. Note that this context is easier to use, since neither computation is needed to obtain the diagonal entries of the matrix N. For more details see [9] and [20].*

Moreover, we have the following practical result:

Proposition 2. *Assume that the algebraic system is split into q blocks, $q \leq \alpha$, corresponding to a coarser subdomain decomposition without overlapping. Then, the parallel asynchronous block relaxation methods associated to this coarser decomposition converge whatever the initial guess V^0 is.*

Proof. The result follows from a direct application of [20]. ■

Remark 9. *As a consequence, it follows that, if the subdomain decomposition associated with α block is a point decomposition (i.e. $\alpha = M$), then the parallel asynchronous block relaxation methods converge for every subdomain decomposition.*

Remark 10. *In the implementation of the algorithm, the number of processors is obviously smaller than the number of blocks in the previous model of parallel iterations. Then several adjacent block components of the discretization matrix, and iterate vectors are processed accordingly by each processor. Such an implementation leads to a more multiplicative behavior of the considered subdomain methods without overlapping.*

Remark 11. *The convergence of synchronous or asynchronous methods can also be established by partial ordering techniques (see [18, 19]). In all cases the convergence follows from the fact that the spatial discretization matrix is an M-matrix.*

3.4. Asynchronous parallel Richardson method

In this subsection, we consider the solution of a problem of option pricing defined by a diffusion operator and associated to an optimization problem with constraints on the solution; then, we will denote by $W \equiv U$ the solution of such a problem. Let K be a closed convex set defined by an analogous way as previously shown

$$K = \{U \mid U \geq \bar{\Phi} \text{ everywhere in } E\}$$

where once again $\bar{\Phi}$ is the discretized *obstacle function*. In fact, after a change of variables and when the formulation of the model problem is derived from

the diffusion operator, this problem is formulated as the following constrained optimization problem

$$\begin{cases} \text{Find } U^* \in K \text{ such that} \\ \forall W \in K, J(U^*) \leq J(W) \end{cases}$$

where the cost function is given by

$$J(U) = \frac{1}{2}\langle \mathcal{A}.U\,, U\rangle - \langle G\,, U\rangle$$

and in which $\langle .\,, .\rangle$ denotes the scalar product in E and $\mathcal{A} = A + \frac{1}{k}I$ is assumed to be symmetric positive definite, according to the properties of the diffusion operator. Classically, it is proved (see [10]) that the discretized solution U^* of the constrained optimization problem is the solution of the discrete complementary problem at each time step

$$\begin{cases} \text{Find } U^* \in \mathbb{R}^M \text{ such that} \\ (A + \frac{1}{k}I)U^* - G \geq 0, U^* \geq \bar{\Phi}, \\ ((A + \frac{1}{k}I)U^* - G)^T(U^* - \bar{\Phi}) = 0, \end{cases} \tag{26}$$

problem equivalent to the discrete variational inequality (11) for every boundary conditions considered on $\partial\Omega$ for the continuous problem.

Owing to the great size of such a system, in order to reduce computation time, the former optimization problem can be solved in a numerical way by using a projected parallel asynchronous method on the convex set. More particularly we will consider next an asynchronous parallel adaptation of the Richardson projected method [21].

For any $\delta \in \mathbb{R}, \delta > 0$, let a fixed point mapping F_δ be defined by

$$\forall W \in E, F_\delta(W) = P_K(W - \delta(\mathcal{A}.W - G)) \tag{27}$$

which can also be written $F_\delta(W) = (F_{1,\delta}(W), \ldots, F_{\alpha,\delta}(W))$ in the following way

$$\forall W \in E, F_{i,\delta}(W) = P_{K_i}(W_i - \delta(\mathcal{A}_i.W - G_i))$$

Then, we can state the following convergence result of the synchronous and asynchronous parallel Richardson method.

Proposition 3. *Assume that (19), (20) and (21) hold. Then there exists a value $\delta_0 > 0$, such that $\forall \delta \in]0, \delta_0[$, the synchronous and the asynchronous iterations defined by (13), (14) and (15), associated with the fixed point mapping F_δ defined by (27), converge to the limit U^* for every initial guess U^0, U^* being the unique solution of problem (26).*

Proof. First of all, note that the projector P_K is a contracting operator. Thus, the problem is to find the set of real numbers δ such that the mapping F_δ is contracting. According to the result of [17], assume that the space E is normed by (25). Let $U \in K$ and $\epsilon = \tilde{U} - U^*$, such that, according to the previous notations, $\epsilon_i = \tilde{U}_i - U_i^*$; let

$$B_i = \frac{1}{n_{i,i}} \frac{|\epsilon_i - \delta \mathcal{A}_i.\epsilon|_i^2}{\Theta_i^2} = \frac{1}{n_{i,i}} \frac{|\epsilon_i|_i^2 - 2\delta \langle \mathcal{A}_i.\epsilon\,, \epsilon_i \rangle + \delta^2 |\mathcal{A}_i.\epsilon|_i^2}{\Theta_i^2}.$$

Let $\mathcal{A}_i.\epsilon = \mathcal{A}_{i,i}.\epsilon_i + \sum_{j \neq i} \mathcal{A}_{i,j}.\epsilon_j$. Since $\mathcal{A}_i \in \mathcal{L}(E, E_i)$, $\exists M_i \geq 0$, $\forall \epsilon \in E$, $|\mathcal{A}_i.\epsilon|_i \leq M_i \|\epsilon\|$. Moreover, according to (19) and (20) the above relation can be overestimated by

$$B_i \leq \frac{1}{n_{i,i}} \frac{|\epsilon_i|_i^2 - 2\delta \sum_{j=1}^{\alpha} n_{i,j} |\epsilon_i|_i |\epsilon_j|_j + \delta^2 M^2 \|\epsilon\|^2}{\Theta_i^2},$$

where $M = \max_{1 \leq i \leq \alpha} M_i$. Then

$$B_i \leq \left(\frac{1}{n_{i,i}} - 2\delta \right) \frac{|\epsilon_i|_i^2}{\Theta_i^2} - 2\delta \left(\sum_{j \neq i} \frac{n_{i,j}}{n_{i,i}} \Theta_j \frac{|\epsilon_j|_j}{\Theta_j} \right) \frac{|\epsilon_i|_i}{\Theta_i^2} + \frac{\delta^2 M^2}{n_{i,i}} \frac{\|\epsilon\|^2}{\Theta_i^2}.$$

So according to (23),

$$B_i \leq \left(\frac{1 - 2\delta\, n_{i,i}}{n_{i,i}} \right) \frac{|\epsilon_i|_i^2}{\Theta_i^2} + 2\delta\, \nu \left(\max_{1 \leq j \leq \alpha} \frac{|\epsilon_j|_j}{\Theta_j} \right) \frac{|\epsilon_i|_i}{\Theta_i} + \frac{\delta^2 M^2}{n_{i,i}} \frac{\|\epsilon\|^2}{\Theta_i^2}.$$

Then

$$B_i \leq \frac{1}{n_{i,i}} \left((1 - 2\delta\, n_{i,i}(1 - \nu)) \|\epsilon\|_{\nu,\Theta}^2 + \frac{\delta^2 M^2}{\Theta_i^2} \|\epsilon\|^2 \right).$$

Let

$$\underline{\Theta} = \min_{1 \leq i \leq \alpha} \Theta_i, \quad \text{and } \underline{n} = \min_{1 \leq i \leq \alpha} n_{i,i}.$$

As $\|\epsilon\|^2 = \sum_{i=1}^{\alpha} \Theta_i^2 \frac{|\epsilon_i|_i^2}{\Theta_i^2} \leq |\Theta|_2^2 \|\epsilon\|_{\nu,\Theta}^2$, where $|\,.\,|_2$ denotes the Euclidean norm in $\mathbb{R}^\alpha$, we finally have

$$\|\epsilon - \delta A.\epsilon\|_{\nu,\Theta}^2 \leq \left(1 - 2\underline{n}(1 - \nu)\delta + M^2 \frac{|\Theta|_2^2}{\underline{\Theta}^2} \delta^2 \right) \|\epsilon\|_{\nu,\Theta}^2.$$

A suitable value for δ is obtained when

$$(1 - 2\underline{n}(1-\nu)\delta + M^2 \frac{|\Theta|_2^2}{\underline{\Theta}^2}\delta^2) < 1.$$

Then, δ being a positive real number, the considered algorithm converges if $\delta \in]0, \delta_0[$, where

$$\delta_0 = \frac{2\underline{n}(1-\nu)\underline{\Theta}^2}{M^2|\Theta|_2^2}.$$

■

Remark 12. *When the convergence is analyzed by contraction techniques, then according to [9, 17, 20] the convergence is purely linear for both Richardson and block relaxation asynchronous algorithms; indeed, in these previous works such results are obtained by using weighted uniform norms and are still valid for all equivalent norms in a finite dimensional space.*

4. Parallel experiments

In the following, we describe the parallel implementation on GPU clusters of the solutions presented in the previous sections for solving the American option derivatives. First, the hardware and software of the GPUs are presented. Then, some experiments are discussed and analyzed.

4.1. GPU architecture

A GPU is viewed as an *accelerator* for the data-parallel and intensive arithmetic computations. It draws its computing power from the parallel nature of its hardware and software architectures. A GPU is composed of hundreds of *Streaming Processors* (SPs) organized in several blocks called *Streaming Multiprocessors* (SMs). It also has a memory hierarchy. It has a private read-write *local memory* per SP, fast *shared memory* and read-only *constant* and *texture* caches per SM and a read-write *global memory* shared by all its SPs [22].

On a CPU equipped with a GPU, all the data-parallel and intensive functions of an application running on the CPU are off-loaded onto the GPU in order to accelerate their computations. A similar data-parallel function is executed on a GPU as a *kernel* by thousands or even millions of parallel threads, grouped together as a grid of thread blocks. Therefore, each SM of the GPU executes one or more thread blocks in SIMD fashion (Single

Instruction, Multiple Data) and in turn each SP of a GPU SM runs one or more threads within a block in SIMT fashion (Single Instruction, Multiple threads). Indeed at any given clock cycle, the threads execute the same instruction of a kernel, but each of them operates on different data.

GPUs only work on data filled in their global memories and the final results of their kernel executions must be communicated to their CPUs. Hence, the data must be transferred *in* and *out* of the GPU. However, the speed of memory copy between the GPU and the CPU is slower than the memory bandwidths of the GPU memories and, thus, it dramatically affects the performances of GPU computations. Accordingly, it is necessary to limit data transfers between the GPU and its CPU during the computations.

4.2. GPU implementations

Algorithm 1 defines the main key points for solving non linear systems derived from the discretization of the obstacle problems in a three-dimensional domain, where A, G and U are, respectively, the discretization matrix, the right-hand side and the vector solution.

Algorithm 1: Algorithm for solving non linear systems of the obstacle problem on GPUs

Input: A (matrix), G (right-hand side), ε (error tolerance threshold), $MaxRelax$ (maximum number of relaxations), $NbSteps$ (number of time steps)

Output: U (solution vector)

```
1 Initialization of the parameters of the obstacle problem;
2 Allocate and fill the data in the global memory GPU;
3 for (i = 1) to NbSteps do
4 |   G ← (1/k)U + F;
5 |   Solve(A, U, G, ε, MaxRelax);
6 end
7 Copy results back from GPU memory;
```

After the initialization step, all data of the obstacle problem to be solved must be copied from the CPU memory to the GPU global memory, because the GPUs only work on the data filled in their memories. Next, the algorithm uses $NbSteps$ time steps for solving the obstacle problem, by using an iterative method defined in the function $Solve()$ in line 5. The iterative

methods used in this algorithm are the projected Richardson method or the projected block relaxation method, that are both adapted to GPUs. At every time step, the initial guess for the iterative solver is set to the solution found at the previous time step. Moreover, the right-hand side, G, is computed as follows (formula (10)):

$$G = \frac{1}{k} \cdot U^{prec} + F,$$

where k is the time step, U^{prec} is the solution computed in the previous time step and each element, $f(x, y, z)$, of the vector F is computed as follows:

$$f(x, y, z) = \cos(2\pi x) \cdot \cos(4\pi y) \cdot \cos(6\pi z).$$

The iterative solver terminates its computations when the error tolerance threshold, ε, and/or the maximum number of relaxations, $MaxRelax$, have been achieved. Finally, the solution of the obstacle problem is copied back from the GPU global memory to the CPU memory. We use the communication functions of the CUBLAS library (CUDA Basic Linear Algebra Subroutines) for the memory allocations in the GPU (`cublasAlloc()`) and the data transfers from the CPU memory to the GPU memory (`cublasSetVector()`) or from the GPU memory to the CPU memory (`cublasGetVector()`).

The projected Richardson solver and the projected block relaxation solver are iterative methods for solving algebraic systems. They are based on arithmetic vector operations that are easy to implement on parallel computers and, thus, on GPUs. Indeed, the GPU executes the vector operations as kernels and the CPU executes the sequential operations, launches the kernels and supplies the GPU with data. Algorithm 2 shows the main key points of both iterative solvers. All the data-parallel arithmetic operations inside the main loop (`repeat ... until(...)`) are executed as kernels by the GPU. We exploit the functions of the CUBLAS library to implement some vector operations on the GPU. We use the following functions:

- `cublasDcopy()` for the data copy of a vector to another vector in the GPU memory (line 3 in Algorithm 2),
- `cublasDaxpy()` to compute the error vector (line 5) and,
- `cublasDnrm2()` to compute the Euclidean norm of the error vector (line 6).

The main kernels of the function `Computation_New_Vector_Components()` (line 4 in Algorithm 2) are those of the matrix-vector multiplication, AU, and

Algorithm 2: A global algorithm of the iterative solvers

Input: A (matrix), G (right-hand side), ε (error tolerance threshold), $MaxRelax$ (maximum number of relaxations)

Output: U (solution vector)

```
1 p ← 0;
2 repeat
3 |   tmp ← U;
4 |   Computation_New_Vector_Components(A, G, U);
5 |   tmp ← tmp − U;
6 |   ρ ← ‖tmp‖_2;
7 |   p ← p + 1;
8 until (ρ < ε) or (p ≥ MaxRelax) ;
```

the vector components updates of U. However, their implementations on a GPU for the projected Richardson solver and the projected block relaxation solver differ in the computing methodology of the iterate solution vector U, such that:

- The projected Richardson method is implemented as a *point-based iteration* and uses the vector updates of the *Jacobi's method*.
- The projected block relaxation method is implemented as a *block-based iteration* and uses the vector block component updates of the *Gauss-Seidel's method*.

The matrix A is a block matrix defined in a three-dimensional domain. It is composed of seven constant coefficients that are called in this paper: $Center$, $West$, $East$, $South$, $North$, $Rear$ and $Front$. Figure 1 shows the positions of these constant coefficients of the matrix A in the three-dimensional domain. The iterations of the projected Richardson method, based on the Jacobi's method, in a three-dimensional domain are defined as follows:

$$\begin{aligned} u^{p+1}(x,y,z) \;=\; & \tfrac{1}{Center}(g(x,y,z) - (Center \cdot u^p(x,y,z) + \\ & West \cdot u^p(x-h,y,z) + East \cdot u^p(x+h,y,z) + \\ & South \cdot u^p(x,y-h,z) + North \cdot u^p(x,y+h,z) + \\ & Rear \cdot u^p(x,y,z-h) + Front \cdot u^p(x,y,z+h))), \end{aligned} \tag{28}$$

where $u^p(x,y,z)$ is an element of the solution vector U computed at the relaxation p and $g(x,y,z)$ is a vector element of the right-hand side G.

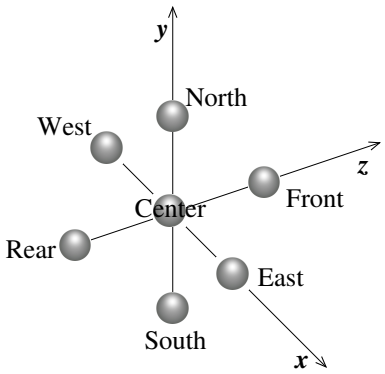

Figure 1: Matrix constant coefficients in a three-dimensional domain.

Figure 2 shows the GPU implementations of both kernels: the matrix-vector multiplication (`MV_Multiplication()`) and the vector components updates (`Vector_Updates()`) of the projected Richardson solver. First, since this method is implemented as a point iteration, each kernel is executed by a large number of GPU threads such that each thread is in charge of the computation of one component of the iterate vector U (see Figure 3). Second, this method uses the vector updates of the Jacobi's method, which means that each thread computes the new value of its component, u_i^{p+1}, independently from the new values, u_j^{p+1} for $j \neq i$, of those computed in parallel by other threads at the same relaxation $p+1$. This methodology allows the maximization of the parallel execution between the GPU streaming processors and, thus, gives good performances for a kernel execution on a GPU. We set the size of a thread block, $Threads$, to 512 threads and the number of thread blocks of a kernel, $Blocks$, is computed so as each GPU thread is in charge of one vector element:

$$Blocks = \frac{NX \cdot NY \cdot NZ + Threads - 1}{Threads},$$

where NX, NY and NZ are the numbers of the vector elements on the coordinate axes x, y and z, respectively.

We can notice from the formula (28) that the computation of a vector element $u^{p+1}(x, y, z)$, by a thread at relaxation $p+1$, requires seven vector elements computed at the previous relaxation p: two vector elements in each dimension plus the vector element at the intersection of the three axises x,

```
/* Kernel of the matrix-vector multiplication */
__global__ void MV_Multiplication (int n, double* U, double* Y)
{ int tid = blockIdx.x * blockDim.x + threadIdx.x; //thread ID
  double sum;
  if(tid < n){
    int x = tid % NX;        //x-coordinate
    int y = (tid / NX) % NY; //y-coordinate
    int z = tid / (NX * NY); //z-coordinate
    sum = Center * fetch_double(U, tid);
    if(x != 0)      sum += West  * fetch_double(U, tid-1);
    if(x != (NX-1)) sum += East  * fetch_double(U, tid+1);
    if(y != 0)      sum += South * fetch_double(U, tid-NX);
    if(y != (NY-1)) sum += North * fetch_double(U, tid+NX);
    if(z != 0)      sum += Rear  * fetch_double(U, tid-NX*NY);
    if(z != (NZ-1)) sum += Front * fetch_double(U, tid+NX*NY);
    Y[tid] = sum;
  }
}
/* Kernel of the vector components updates */
__global__ void Vector_Updates(int n, double* G, double* Y, double* U)
{ int tid = blockIdx.x * blockDim.x + threadIdx.x; //thread ID
  double var;
  if(tid < n){
     var = (G[tid] - Y[tid]) / Center + fetch_double(U, tid);
     if(var < 0) var = 0; //projection
     U[tid] = var;
  }
}
/* Function to be executed by the CPU */
void Computation_New_Vector_Components(double* A, double* G, double* U)
{ int nnn = NX * NY * NZ;                    //number of vector elements
  int Threads = 512;                         //size of a thread block
  int Blocks = (nnn + Threads - 1) / Threads; //number of thread blocks
  double* Y;
  //Matrix coefficients filled in the constant memory:
  //Center, West, East, South, North, Rear, Front
  //vector Elements of U are filled in the texture memory
  //Allocate a GPU memory space for the vector Y
  MV_Multiplication<<<Blocks,Threads>>>(nnn, U, Y);
  Vector_Updates<<<Blocks,Threads>>>(nnn, G, Y, U);
}
```

Figure 2: GPU kernels of the projected Richardson solver

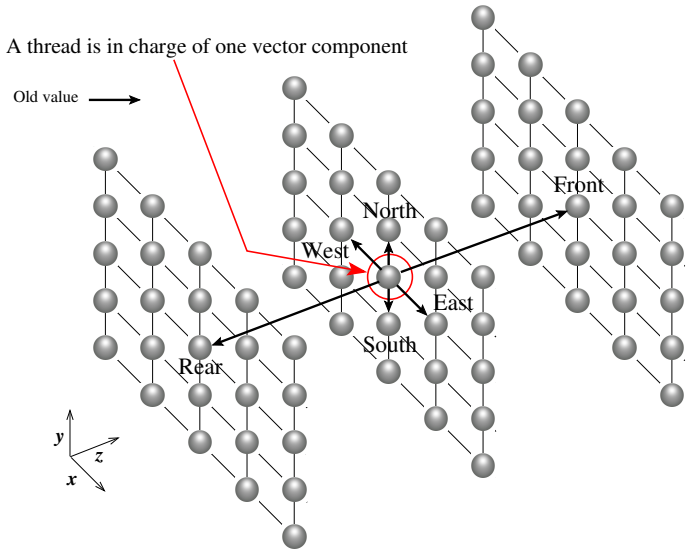

Figure 3: Computation of a vector component with the projected Richardson method.

y and z (see Figure 3). In order to reduce the memory accesses to the high-latency global memory, the element of the vector iterate U are filled in the cached texture memory (see [31]). Obviously, the vector elements of U could be stored in the low-latency shared memories of thread blocks, as is described in [30]. Nevertheless, the fact that the computation of a vector element requires only two elements in each dimension does not allow to maximize the data reuse from the shared memories. Moreover, the computation of the bordering vector elements in each thread block involves more conditional statements in the kernels, in order to fill the required vector elements in the shared memory. The conditional statements inside the kernels may result in performance degradation of GPU based algorithms. So, in the kernel codes (Figure 2), we use the function `fetch_double(vect,i)` to read from the texture memory the ith double-precision element of the vector `vect`. Finally, the seven constant coefficients of matrix A are filled in the cache constant memory.

In contrast, the methodology used by the projected block relaxation solver to compute the iterate vector U does not allow the maximization of the parallel execution on a GPU. First, the block-based methods require the triangular solving of the matrix blocks, for example, along the x axis according to the

```
/* Kernel of the forward matrix-vector multiplication */
__global__ void Forwrd_MV(int n, double* U, double* Y)
{ int tid = blockIdx.x * blockDim.x + threadIdx.x; // thread ID
  if(tid < n){
    int y = (tid / NX) % NY; //y-coordinate
    int z = tid / (NX * NY); //z-coordinate
    double sum = 0;
    if(y != (NY-1)) sum += North * fetch_double(U, tid+NX);
    if(z != (NZ-1)) sum += Front * fetch_double(U, tid+NX*NY);
    Y[tid] = sum;
  }
}
/* Kernel of the backward matrix-vector multiplication */
__global__ void Backwrd_MV(int n, int y, int z, double* U, double* Y)
{ int tid = blockIdx.x * blockDim.x + threadIdx.x; // thread ID
  if(tid < n){
    double sum = 0;
    if(y != 0) sum += South * fetch_double(U, tid-NX);
    if(z != 0) sum += Rear  * fetch_double(U, tid-NX*NY);
    Y[tid] += sum;
  }
}
/* Function to be executed by the CPU */
void Computation_New_Vector_Components(double* B, double* A, double* G, double* U)
{ int i, y, z;
  int nnn = NX * NY * NZ;                    //number of vector elements
  int Threads = 512;                         //size of a thread block
  int Blocks = (nnn + Threads - 1) / Threads; //number of thread blocks
  double* Y;
  //B is a the triangularization of A (triangular matrix)
  //Matrix coefficients of A and B filled in the constant memory
  //vector Elements of U are filled in the texture memory
  //Allocate a GPU memory space for the vector Y
  Forwrd_MV<<<Blocks,Threads>>>(nnn, U, Y);
  Blocks = (NX + Threads - 1) / Threads;
  for(i=0; i<NY*NZ; i++){
    y = i % NY; //y-coordinate
    z = i / NY; //z-coordinate
    Backwrd_MV<<<Blocks,Threads>>>(NX, y, z, &U[i*NX], &Y[i*NX]);
    Back_Solves<<<1,1>>>(B, &G[i*NX], &Y[i*NX], &U[i*NX]);
  }
}
```

Figure 4: GPU kernels of the projected block relaxation solver

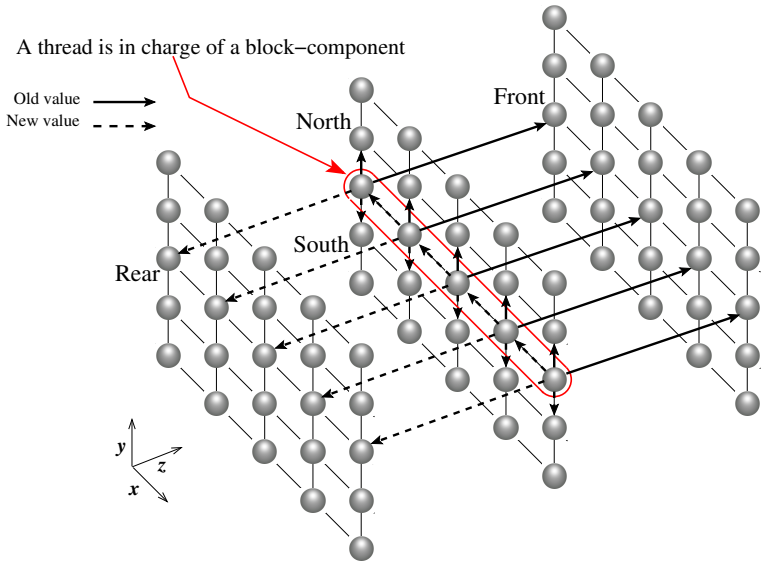

Figure 5: Computation of a vector component with the projected block relaxation method.

numbering of the grid points. Therefore, at the initialization step in Algorithm 1, a triangularization of the largest block of the tridiagonal matrix $\mathcal{A}_{i,i}$, is performed along the x axis. Figure 4 shows the GPU implementations of the kernels of the projected block relaxation solver. So at each relaxation, this solver performs triangular solving by the back substitution method in each block-component (`Back_Solves()`). In this case, each GPU thread is in charge of a block of vector components (see Figure 5) instead of only one component as the projected Richardson method. Moreover, since this solver is based on the Gauss-Seidel's method, the computation of the new value of a block-component U_i^{p+1} involves the old values U_j^p for $j > i$ (`Forwrd_MV()`) and the new values U_k^{p+1} for $k < i$ (`Backwrd_MV()`) of other block-components of the iterate vector U. Consequently, each GPU thread must wait for the computing of the new values of other block-components by other threads before computing the new values of the components of its own block. However, the fact that a thread has to wait for other threads to complete their computations dramatically affects the computation performances of the GPUs. Therefore, the projected block relaxation method performs the triangular solving and the vector updates on the CPU, which involves in each relaxation data transfers between the CPU and the GPU.

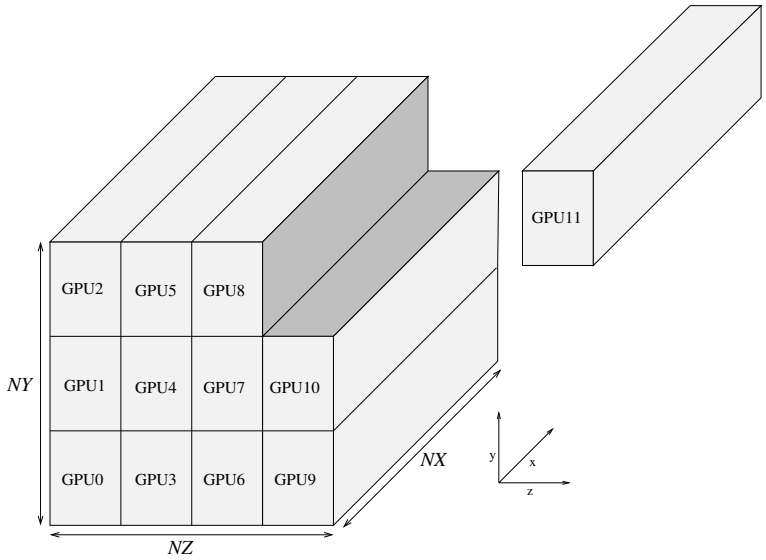

Figure 6: Data partitioning of an obstacle problem among $n = 3 \times 4$ GPUs.

To solve large scale algebraic systems derived from the discretization of the obstacle problems, the algorithms of both methods must be parallelized between several GPUs in order to reduce the elapsed computation time. We develop parallel *synchronous* and *asynchronous* algorithms for each method. Let n be the number of GPUs in the parallel architecture, for example a cluster of GPUs. First, the obstacle problem to be solved must be split in n subproblems, such that each subproblem is affected to one GPU. Indeed, the y and z axes of the three-dimensional domain of the problem are respectively split into ny and nz parts, such that $n = ny \times nz$. Figure 6 shows an example of the data partitioning of an obstacle problem of size $NX \times NY \times NZ$ among twelve GPUs. This parallelepipedic data partitioning of the problem reduces the data exchanges at subdomain boundaries compared to a naive z-axis-wise partitioning. After the decomposition of the problem, all the data of each subproblem are copied from the CPU memory to the GPU global memory, in order to be processed on the GPU.

Then, the same algorithm adapted to GPUs for each method described above is executed in parallel by each GPU on its local data and synchronizations of all computations are performed between neighboring GPUs. For every relaxation p of the method, each computing node (a CPU and its GPU)

performs this following algorithm:

1. Repeat until convergence,
2. Compute the values associated to bordering points shared with neighbors,
3. Copy the shared values associated to bordering points from the GPU memory to the CPU memory,
4. Exchange the values associated to bordering points between the neighboring CPUs,
5. Copy the received values associated to bordering points from the CPU memory to the GPU memory,
6. Compute the local new components.

For the data communications between a CPU and its GPU, we use the synchronous communication routines of the CUBLAS library: `cublasGetVector()` and `cublasSetVector()`. These routines allow the overlap of the data transfer and the kernel execution. However, the data exchanges of the values associated to the bordering points are performed between the neighboring CPUs via MPI communication routines. For the parallel *synchronous* algorithms, we use the MPI subroutine `MPI_sendrecv()` which has a good throughput. In contrast, for the parallel *asynchronous* algorithms, we use the MPI non-blocking communications: `MPI_Issend()`, `MPI_Irecv()` and `MPI_Test()`.

4.3. Experiments

In this section, we discuss the performance behaviors of the projected block relaxation method and the projected Richardson method implemented on CPUs and on GPUs.

We performed a set of experiments on the sequential and the parallel algorithms, for both methods and different sizes. We took into account the execution times and the number of relaxations performed by each method on CPUs and on GPUs.

All experimental results obtained from the simulations are made in double precision data, for a convergence tolerance of the methods set to 10^{-4} and a maximum number of relaxations limited to 10^6 relaxations. Since we were more interested in the comparison of the performance behaviors of these methods on CPUs versus on GPUs, we have limited our experiments to the first three time steps. The numerical values of the obstacle problems input data are those given in Section 3.1 for the numerical solutions.

Pb. size	Method	$Time_{cpu}$	$Time_{gpu}$	Nb. relax.	τ_{rel}	τ_{max}
32^3	Block relaxation	0.09	0.07	54	1.28	1.5
	Richardson	0.16	0.06	184	2.67	
64^3	Block relaxation	3.82	1.15	161	3.32	8.88
	Richardson	10.68	0.43	649	24.84	
128^3	Block relaxation	138.10	29.23	556	4.72	17.30
	Richardson	440.05	7.98	2,418	55.14	
256^3	Block relaxation	5,760.65	834.80	2,015	6.90	24.45
	Richardson	22,551.48	235.61	9,166	95.71	

Table 1: Execution times in seconds of the projected block relaxation method and the projected Richardson method on a CPU core vs. on a Tesla GPU

In Table 1, we report the performances of the sequential algorithm of the projected block relaxation method and that of the projected Richardson method for solving algebraic systems derived from the discretization of the obstacle problems of sizes 32^3, 64^3, 128^3 and 256^3. In the third and fourth columns, we report respectively the execution times of the methods implemented on one core of the Quad-Core Xeon E5530 CPU and those of the same methods implemented on one Nvidia Tesla C1060 GPU. The fifth column shows the total number of relaxations performed by each method to converge. In the sixth and seventh columns are reported the real numbers τ_{rel} and τ_{max} as the ratios between the execution times obtained on the CPU and that obtained on the GPU. Indeed, the ratios τ_{rel} define the relative gains of a method implemented on the GPU compared to the same method implemented on the CPU. In contrast, the ratios τ_{max} define the performance gains obtained in solving the non linear systems with the best method on the GPU (projected Richardson method) compared to solving them with the best method on the CPU (projected block relaxation method).

From the ratios (τ_{rel} and τ_{max}) shown in Table 1, we can see that the methods implemented on the GPU are faster than those implemented on the CPU. This is due to the GPU ability to compute the data-parallel functions faster than its CPU counterpart. However for the different problem sizes of our simulations, the ratios τ_{rel} of the projected block relaxation method are largely inferior to those of the projected Richardson method. We can also notice that the projected block relaxation method is almost 3 times faster than the projected Richardson method on the CPU, while it is 3 times

slower than this method on the GPU even if it converges 4 times faster than the projected Richardson method (see number of relaxations). So this means that a relaxation of the projected block relaxation method runs on the GPU more slowly than that of the projected Richardson method. In fact, the fixed point-based nature and the vector updates of the Jacobi method allow a straightforward and efficient thread-parallelization of the projected Richardson method on the GPU, such that each component of the iterate vector is computed in parallel and independently from other components. Unfortunately this is not the case in the projected block relaxation method. At every relaxation, the triangular solves and the Gauss-Seidel vector updates are performed by the CPU, which is slower than the GPU, and they involve slow data transfers between the GPU and its CPU. In order to improve the performances of the projected block relaxation method in GPUs, we can replace the Gauss-Seidel method by the Jacobi one. Indeed, this last method allows each thread to compute its block-component independently from the block-components of other threads. Nevertheless, the block-based method uses a triangular solving to compute the vector components in each block, which is not easy to parallelize on GPUs. Then, we focus our test experiments on the projected block relaxation solver based on the Gauss-Seidel method, since it is more efficient on the CPUs than the one based on the Jacobi method. Finally, we can see from the ratios τ_{max} of our simulations that solving non linear systems with the best method on the GPU (projected Richardson method) is from twice to 24 times faster than solving them with the best method on the CPU (projected block relaxation method).

In Table 2 and Table 3, we report the execution times and the total number of relaxations performed by the parallel projected block relaxation method and the parallel projected Richardson method for solving large-scale non linear systems derived from the discretization of the obstacle problems of sizes 256^3 and 512^3. We implemented the parallel synchronous and asynchronous versions of each method on a cluster of six Quad-Core Xeon E5530 CPUs (24 CPU cores) and on a cluster of 12 Nvidia Tesla C1060 GPUs. In Table 4 we show the ratios τ_{rel} and τ_{max} between the execution times obtained on the CPU cluster and those obtained on the GPU cluster for both versions of each method.

Table 4 shows that both parallel versions (synchronous and asynchronous) of each method are faster on the GPU cluster than on the CPU cluster. However, as we noticed previously, the performance improvements of the projected fixed point Richardson method on the GPU cluster are better

Pb. size	Method	Synchronous		Asynchronous	
		$Time_{cpu}$	Nb. relax.	$Time_{cpu}$	Nb. relax.
256^3	Block relaxation	137.50	37,080	131.71	38,348
	Richardson	575.225	198,288	539.25	198,613
512^3	Block relaxation	4,814.71	132,600	4,371.59	141,067
	Richardson	19,250.25	750,912	18,160.42	768,186

Table 2: Execution times in seconds of the parallel projected block relaxation method and the parallel Richardson method implemented on a cluster of 24 CPU cores

Pb. size	Method	Synchronous		Asynchronous	
		$Time_{gpu}$	Nb. relax.	$Time_{gpu}$	Nb. relax.
256^3	Block relaxation	66.28	19,548	63.53	20,277
	Richardson	29.69	100,692	20.73	95,265
512^3	Block relaxation	1,825.83	69,960	1,764.79	70,984
	Richardson	602.17	381,300	516.18	348,305

Table 3: Execution times in seconds of the parallel projected block relaxation method and the parallel Richardson method implemented on a cluster of 12 Tesla GPUs

Pb. size	Method	Synchronous		Asynchronous	
		τ_{rel}	τ_{max}	τ_{rel}	τ_{max}
256^3	Block relaxation	2.07	4.63	2.07	6.35
	Richardson	19.37		29.01	
512^3	Block relaxation	2.64	7.99	2.48	8.47
	Richardson	31.97		35.18	

Table 4: Ratios between the elapsed times obtained on a cluster of 24 CPUs cores and those obtained on a cluster of 12 Tesla GPUs

than that of the projected block relaxation method, because this last one involves CPU computations and more data transfers between the GPUs and the CPUs. So, as we can see from tables 2 and 3, the projected block relaxation method remains the fastest solver on the CPU cluster and the projected fixed point Richardson method the fastest one on the GPU cluster. We can also notice that the parallel asynchronous version of each method is as fast as the parallel synchronous one in both GPU and CPU clusters.

5. Conclusion and perspectives

In this paper we have presented the parallel synchronous and asynchronous relaxation methods for the American options derivative problem. Moreover, we have proved the convergence of the parallel subdomain method without overlapping and of the parallel projected Richardson algorithm.

Then, we have described the parallel implementation of these algorithms on GPU clusters. We have performed some experiments comparing the parallel CPU and GPU versions. These experiments lead us to conclude that the iterative methods using data-parallel operations are more efficient on the GPUs than on the CPUs, due to the parallel nature of the hardware and software architectures of the GPUs. Moreover, the best solutions for solving algebraic systems on the CPUs are not necessary well-suited to the GPUs. The block-based iteration and the Gauss-Seidel vector updates, that allow the solvers to achieve fast convergence on the CPUs, are less efficient on the GPUs. In contrast, the point-based iteration and the Jacobi vector updates allow a good thread-parallelization, and thus, provide good performances for the iterative methods on the GPUs even if they do not allow them a fast convergence.

In future work, we plan to perform some experiments on larger GPU clusters. This would probably highlight a better efficiency of the asynchronous version. Moreover it would be interesting to study other parallel synchronous and asynchronous numerical methods like two stages methods and more generally the multisplitting method in geographically distant GPU clusters, and red-black ordering method to speed up the convergence of the Richardson method on the GPU cluster.

Acknowledgment

This paper is based upon work partially supported by the Région de Franche-Comté. We thank the reviewers whose insightful and constructive comments have helped improve this paper significantly.

[1] L. Badea, X.C. Tai and J. Wang, Convergence Rate Analysis of a Multiplicative Schwarz Method for Variational Inequalities, SIAM Journal on Numerical Analysis, 41(3): 1052-1073, 2004.

[2] L. Badea and J. Wang, An Additive Schwarz Method for Variational Inequalities, Mathematics of Computation, 69(232): 1341-1354, 2000.

[3] J.M. Bahi, S. Contassot-Vivier and R. Couturier, Parallel Iterative Algorithms: From Sequential to Grid Computing, Chapman & Hall/CRC, Numerical Analysis & Scientific Computating, 1, 2007.

[4] V. Barbu, Nonlinear Semigroups and Differential Equations in Banach Spaces, Noordhoff International Publishing, 1976.

[5] G. Baudet, Asynchronous Iterative Methods for Multiprocessors, Journal of Association for Computing Machinery (JACM), 25(2): 226-244, 1978.

[6] D. Bertsekas and J. Tsitsiklis, Parallel and Distributed Computation, Numerical Methods, Prentice Hall Englewood Cliffs N.J., 1989.

[7] D. Chazan and W. Miranker, Chaotic Relaxation, Linear Algebra and its Applications, 2(2): 199-222, 1969.

[8] A. Gaikwad and I.M. Toke, Parallel Iterative Linear Solvers on GPU: A Financial Engineering Case, Parallel, Distributed, and Network-Based Processing, Euromicro Conference, 0: 607-614, 2010.

[9] L. Giraud and P. Spiteri, Résolution Parallèle de Problèmes aux Limites Non Linéaires, M2 AN, 25: 579-606, 1991.

[10] R. Glowinski, J.L. Lionsa and R. Tremolieres, Analyse Numérique des Inéquations Variationnelles, DUNOD, tome 1 and 2, 1976.

[11] P. Jaillet, D. Lamberton and B. Lapeyre, Variational Inequalities and the Pricing of American Option, Acta Applicandae Mathematicae, 21(3): 263-289, 1990.

[12] Y.A. Kuznetsov, P. Neittaanmaki and P. Tarvainem, Schwarz Methods for Obstacle Problems with Convection-Diffusion Operators, In Proceedings of Domain Decomposition Methods in Scientifical and Engineering Computing, Edited by D.E. Keyes and J.C. Xu, AMS, 251-256, 1995.

[13] Y.A. Kuznetsov, P. Neittaanmaki and P. Tarvainen, Block Relaxation Methods for Algebraic Obstacle Problems with M-matrices, East-West J. Numer. Math., 2(1):75-89, 1994.

[14] C. Li, J. Zeng and S. Zhou, Convergence Analysis of Generalized Schwarz Algorithms for Solving Obstacle Problems with T-monotone Operator, Computers & Mathematics with Applications, 48(3-4): 373-386, 2004.

[15] P.L. Lions and B. Mercier, Approximation Numérique des Equations de Hamilton-Jacobi-Bellman, R.A.I.R.O. Analyse Numérique, 14: 369-393, 1980.

[16] A. Maringanti, V. Athavale and S.B. Patkar, Acceleration of Conjugate Gradient Method for Circuit Simulation using CUDA, 16th International Conference on High Performance Computing (HiPC 2009), 438-444, 2009.

[17] J.C. Miellou, Algorithmes de Relaxation Chaotique à Retards, RAIRO Analyse Numérique, R1: 55-82, 1975.

[18] J.C. Miellou, Asynchronous Iterations and Order Intervals, Parallel Algorithms, M. Cosnard and al. eds.: North-Holland-Amsterdam, 85-96, 1986.

[19] J.C. Miellou, D. El Baz and P. Spiteri, A New Class of Asynchronous Iterative Algorithms with Order Interval, Mathematics of Computation, 67(221): 237-255, 1998.

[20] J.C. Miellou and P. Spiteri, Un Critère de Convergence pour des Méthodes Générales de Point Fixe, M2AN, 19: 645-669, 1985.

[21] J.C. Miellou and P. Spiteri, Two Criteria for the Convergence of Asynchronous Iterations, Computers and Computing, P. Chenin and al. ed., Wiley Masson, Paris, 91-95, 1985.

[22] Nvidia Corporation, NVIDIA CUDA C Programming Guide, Version 3.2, 2010.

[23] P. Spiteri, A New Characterization of M-matrices and H-matrices, BIT Numerical Mathematics, Springer, 43(5): 1019-1032, 2003.

[24] X.C. Tai, Convergence Rate Analysis of Domain Decomposition Methods for Obstacle Problems, East-West J. Numer. Anal., 9(3): 233-252, 2001.

[25] X.C. Tai and P. Tseng, Convergence Rate Analysis of an Asynchronous Space Decomposition Method for Convex Minimization, Mathematics of Computation, 71(239): 1105-1135, 2001.

[26] X.C. Tai, Rate of Convergence for Some Constraint Decomposition Methods for Nonlinear Variational Inequalities, Numerische Mathematik, 93(4): 755-786, 2003.

[27] R. Varga, Matrix Iterative Analysis, Prentice Hall, 1962.

[28] P. Wilmott, J. Dewyne and S. Howison, Option Pricing-Mathematical Models and Computation, Oxford financial press, 1993.

[29] Y. Zhao, Lattice Boltzmann Based PDE Solver on the GPU, The Visual Computer: International Journal of Computer Graphics, 24(5): 323-333, 2008.

[30] P. Micikevicius, 3D Finite Difference Computation on GPUs using CUDA, Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units, 79–84, 2009.

[31] A. Leist, D. P. Playne and K. A. Hawick, Exploiting Graphical Processing Units for Data-parallel Scientific Applications, Concurrency and Computation: Practice and Experience, 21(18): 2400–2437, 2009.