Asynchronous iterations of Parareal algorithm for option pricing models

Frédéric Magoulès, Guillaume Gbikpi-Benissan, Qinmeng Zou

Asynchronous Iterations of Parareal Algorithm for Option Pricing Models

Frédéric Magoulès* Guillaume Gbikpi-Benissan† Qinmeng Zou‡

Abstract

Spatial domain decomposition method has been largely investigated in the last several decades, while time decomposition seems against intuition, which is not as popular as the former. However, there are still many attractive methods being proposed, especially the parareal algorithm, which shows both theoretical and experimental efficiency in the context of parallel computing. In this paper, we present an original model of asynchronous variant based on parareal scheme, applied to the European option pricing problem. Some numerical experiments are given to illustrate the convergence performance and computational efficiency of such method.

Keywords: Parallel computing; asynchronous iterations; parareal method; European options; domain decomposition; time-dependent problems

1 Introduction

Today's dominating high-performance computer architecture is parallel. Computer Assisted Engineering (CAE) generally lead to problems that are not naturally parallelizable. Strong effort was done during the last years to propose high-performance decomposition domain methods (DDM) that support scalability and reach high rates of speedup and efficiency.

Since about two decades, people have also tried to propose parallel-in-time algorithms. Although it can be appear as unnatural because of time-line "orientation", some attempts did succeed for particular problems or equations, inviting people to look forward and continue research in this direction. Multiple shooting methods [12, 17] for example were proposed to allow for parallel computations of initial-value problems of differential equations. Another approach is the time decomposition method originally introduced by researchers from the multi-grid field [29, 32] and applied to solve PDEs. Finally, people also tried to apply spatial domain decomposition methods for time dependent problems. The so-called Waveform relaxation methods (see, e.g., [27]) distribute the computation on parallel computers by partitioning the system into subsystems and then use a Picard iteration to compute the global solution. The major flaw of these methods is their low convergence rate. To make them efficient, one needs to use time-dependent transmission conditions adapted to the underlying problem [26].

The recent parareal scheme (resp. PITA algorithm) proposed in [33] (resp. [22]) follows both multiple shooting and multi-grid approaches. Two levels of grid in time are considered in order to split the time domain into subdomains. A prediction of the solution is parallel computed on the fine grid in time. Then at each end-boundary of time subdomains, the solution makes a jump with the previous initial boundary value (IBV) to the next time subdomain. A correction of the IBV for the next fine grid iterate is then computed on the coarse grid in time. Ref. [22] shows that the method converges at least in a finite number of iterations, due to the propagation of the fine time grid solution as at each iteration k, the

*CentraleSupélec, Université Paris-Saclay, France (correspondence, frederic.magoules@hotmail.com).
†CentraleSupélec, Université Paris-Saclay, France (guibenissan@gmail.com).
‡CentraleSupélec, Université Paris-Saclay, France (zouqinmeng@gmail.com).

IBV of the kth time subdomain is exact. Generalizations of the parareal scheme then were proposed by introducing a wider class of coarse solvers that are not specifically defined on coarser sub-grids in time. Coarse solvers can be simpler physical models where fine physics is approximated or simply skipped.

On the other hand, an efficient computational scheme has been proposed to confront the drawbacks of the classical parallel methods, which is called asynchronous iterative scheme. The primordial idea was put forward by Chazan and Miranker [13] for the solving of linear systems, where a necessary and sufficient convergence condition was derived. Several extensions were then developed based on the operator theory applied to the nonlinear problems [44, 7, 21, 46]. Furthermore, we cite also the work of Bertsekas and Tsitsiklis [8, 9] for the general theories of asynchronous iterations, which leads to a great deal of striking applications (see, e.g., [41, 40]). In [9], the asynchronous scheme is designed to be divided into two types, which are called totally asynchronous iterations and partially asynchronous iterations. The major difference consists in whether the bounded assumption being used for the communication time, where the negative answer was first formalized in [7]. Recently, a brand-new scheme called asynchronous iterations with flexible communication was proposed in [20, 45, 24], which was continually under investigation in the following periods [25, 19]. The key idea behind this scheme is that items are sent to the other processors as soon as it is obtained, regardless of the local completion. Accordingly, such scheme is built upon the two-stage model, which was introduced in [23], or the partial update situation, see [19] for further information. An attempt to present all the theoretical and practical efforts is beyond the scope of our paper, thus we refer the reader to [25, 3] for a broader discussion.

In this paper, we concentrate on a modified parareal algorithm which will be enhanced by the asynchronous iterative scheme with flexible communication. In Section 2, we formalize an option pricing model which will be considered to the end. Section 3 gives the details of asynchronous parareal algorithm. Then, we implement the asynchronous solver in Section 4, where we present the programming trick using an advanced asynchronous communication library. Finally, Section 5 is devoted to the numerical experiments, and concluding remarks are presented in Section 6.

2 Problem Formulation

2.1 Overview

An option is a contract that gives its owner the right to trade in a fixed number of shares of the underlying asset at a fixed price at any time on or before a given date, which is a prominent form of financial derivatives that have been derived from other financial instruments, mainly used for hedging and arbitrage. The right to buy a security is called a call option, whereas the right to sell is called a put option.

Option pricing remained a frustrating problem that was obscure to solve, until the revolutionary advent of Black-Scholes model. The pioneering work was done by Black, Merton, and Scholes [10, 42, 43] in the early 1970s. In quick succession, Cox and Ross [15] proposed the risk neutral pricing theory, which leads to the famous martingale pricing theory [28] by Harrison and Kreps. Meanwhile, Cox, Ross, and Rubinstein simplified the Black-Scholes model, giving birth to the binomial options pricing model [16]. More recently, some stochastic volatility option models have been proposed to better simulate the real volatility in the financial market (see, e.g., [30, 18]).

To the end, we will investigate the original Black-Scholes equation, to solve the European call option pricing problem. Consider the following equation

$$\frac{\partial V}{\partial t} + rS\frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} = rV, \tag{1}$$

where V is the price of the option as a function of underlying asset price S and time t and the parameters are volatility σ and risk-free interest rate r, with the final and boundary

conditions given by

$$\begin{cases} V(S,t) = \max(S-E,0), & t = T,\ S \in [0,+\infty), \\ V(S,t) = 0, & t \in [0,T],\ S = 0, \\ V(S,t) \sim S - Ee^{-r(T-t)}, & t \in [0,T],\ S \to +\infty, \end{cases}$$

where T is the time to maturity, and E is the exercise price. There are many assumptions to be held: (i) the underlying asset follows geometric Brownian motion with drift rate μ and volatility σ constant, (ii) the underlying asset does not pay dividends, (iii) the risk-free interest rate r is constant, (iv) there are no transaction costs or taxes, (v) trading in assets is a continuous process, (vi) the short selling is permitted, (vii) there are no arbitrage opportunities.

2.2 Derivation of the Black-Scholes Equation

To derive this outstanding equation, it is necessary to establish a model for the underlying asset price. Economic theory suggest that a stock price follows a generalized Wiener process

$$\Delta S = \nu S \Delta t + \sigma S \Delta Z.$$

The right hand side is composed of two parts. The first contains a drift rate ν, which depicts the trend of stock price, whereas as the second item is a standard Wiener process, also known as Brownian motion, with a volatility parameter σ, describing the standard deviation of the stock's returns. The stochastic term Z takes on the expression

$$\Delta Z = \epsilon \sqrt{\Delta t},$$

where ϵ is a standard normal distribution variable, while stock price follows the lognormal distribution. In the limit, as $\Delta t \to 0$, this becomes

$$dS = \nu S dt + \sigma S dZ. \tag{2}$$

Therefore, the option price is a function of stock price and time. From Itô's lemma [31], it follows the process

$$dV = (\nu S \frac{\partial V}{\partial S} + \frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2})dt + \sigma S \frac{\partial V}{\partial S} dZ. \tag{3}$$

Since a square-root term $\sqrt{\Delta t}$ exists in the stochastic process, the second-order term is kept during the Taylor series expansions. It is seen that solving the equation seems obscure in view of the stochastic term dZ. The main conceptual idea proposed by Black and Scholes lies in the construction of a portfolio consisting of underlying stock and option that is instantaneously risk-less. Let Π be the value of portfolio consisting of one short position derivative and $\frac{\partial C}{\partial S}$ units of the underlying asset. Then, the value of the portfolio is given by

$$\Pi = -V + \frac{\partial V}{\partial S} S. \tag{4}$$

Hence, the instantaneous change in the portfolio becomes

$$d\Pi = -dC + \frac{\partial C}{\partial S} dS.$$

Substituting (2) and (3) yields

$$d\Pi = (-\frac{\partial C}{\partial t} - \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 C}{\partial S^2})dt. \tag{5}$$

The change of the portfolio value is not dependent on dZ, which means that the portfolio is risk-less during the time interval dt. As mentioned before, there are no arbitrage opportunities in the financial market, so that the return of this portfolio must be equal to other risk-free securities. Thus

$$d\Pi = r\Pi dt. \tag{6}$$

Substituting (4) and (5) into (6) yields

$$\frac{\partial V}{\partial t} + rS\frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} - rV = 0,$$

where we obtain the original Black-Scholes equation (1), which involves adequate conditions to reach the final solution.

2.3 Transformation into the Heat Equation

It is seen that the Black-Scholes partial differential equation (1) is a parabolic linear equation, while the heat equation leads to a simpler one. Accordingly, the transformation from the original to the latter can simplify the solving process. A change of variables is given as

$$S = Ee^x, \quad t = T - \frac{2\tau}{\sigma^2}, \quad V = Ev.$$

Substituting into the Black-Scholes equation (1) gives

$$\frac{\partial v}{\partial \tau} = (\kappa - 1)\frac{\partial v}{\partial x} + \frac{\partial^2 v}{\partial x^2} - \kappa v,$$

where $\kappa = \frac{2r}{\sigma^2}$. Setting

$$\alpha = \frac{1}{2}(\kappa - 1), \quad \beta = \frac{1}{2}(\kappa + 1), \quad v = e^{-\alpha x - \beta^2 \tau} u,$$

then gives the heat equation in an infinite interval

$$\frac{\partial u}{\partial \tau} = \frac{\partial^2 u}{\partial x^2}, \quad \tau \in [0, \frac{T\sigma^2}{2}], \ x \in \mathbb{R}. \tag{7}$$

We notice that the heat equation is forward parabolic, whereas the Black-Scholes equation is backward. In particular, the initial and boundary conditions become

$$\begin{cases} u(x,\tau) = \max(e^{\beta x} - e^{\alpha x}, 0), & \tau = 0, \ x \in \mathbb{R}, \\ u(x,\tau) \sim 0, & \tau \in [0, \frac{T\sigma^2}{2}], \ x \to \pm\infty. \end{cases}$$

Unfortunately, we address the issue that such infinity conditions can not be applied directly to the discrete applications. A suitably large boundary instead of the original boundary is supposed to be considered. Accordingly, we choose the precise boundaries as following

$$\begin{cases} x^- = \min(x_0, 0) - \log(4), \\ x^+ = \max(x_0, 0) + \log(4), \end{cases}$$

such that

$$\begin{cases} u(x,\tau) = 0, & \tau \in [0, \frac{T\sigma^2}{2}], \ x = x^-, \\ u(x,\tau) = e^{\beta x + \beta^2 \tau} - e^{\alpha x + \alpha^2 \tau}, & \tau \in [0, \frac{T\sigma^2}{2}], \ x = x^+. \end{cases}$$

We then notice that the equation (7) can be solved employing appropriate time and space discretization.

2.4 Black-Scholes Pricing Formula

Without numerical tools, we can also deduce the solution of (7), called the Black-Scholes formula, by some analytic procedures. In the case of heat equation, the fundamental solution is

$$\mathcal{G}(x,\tau) = \frac{1}{\sqrt{4\pi\tau}} \exp(-\frac{x^2}{4\tau}). \tag{8}$$

In particular, the general solution of the heat equation can be presented by the convolution integral

$$u(x,\tau) = \int_{-\infty}^{+\infty} \mathcal{G}(x-y,\tau)u_0(y)dy,$$

where $u(x,0) = u_0(x)$ with $x \in \mathbb{R}$ and $\tau \geq 0$. Substituting (8) yields

$$u(x,\tau) = \frac{1}{\sqrt{4\pi\tau}} \int_{-\infty}^{+\infty} \exp(-\frac{(x-y)^2}{4\tau})u_0(y)dy.$$

We note that applying the initial condition into such eqaution still involves some brute force. To summarize, the final closed-form solution is

$$V(S,t) = S\mathcal{N}(d_1) - Ee^{-r(T-t)}\mathcal{N}(d_2),$$

where $\mathcal{N}(x)$ is the cumulative distribution function of the standard normal distribution

$$\mathcal{N}(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{1}{2}z^2} dz,$$

with the parameters d_1 and d_2 shown as below

$$d_1 = \frac{\ln\frac{S}{E} + (r + \frac{\sigma^2}{2})(T-t)}{\sigma\sqrt{T-t}}, \quad d_2 = \frac{\ln\frac{S}{E} + (r - \frac{\sigma^2}{2})(T-t)}{\sigma\sqrt{T-t}}.$$

We notice that $d_2 = d_1 - \sigma\sqrt{T-t}$.

3 Asynchronous Parareal Algorithm

3.1 Asynchronous Iteration

Let E be a product space with $E = E_1 \times \cdots \times E_p$ and let $f : E \to E$ be the function defined by $f_i : E \to E_i$ with

$$f(x) = [f_1(x), \ldots, f_p(x)], \quad x = (x_1, \ldots, x_p) \in E.$$

Let $k \in \mathbb{N}$, $P^k \subseteq \{1, \ldots, p\}$ and $P^k \notin \emptyset$. We assume that

$$\forall i \in \{1, \ldots, p\}, \quad \text{card}\{k \in \mathbb{N} \mid i \in P^k\} = +\infty \tag{9}$$

For $i = 1, \ldots, p$ and $j = 1, \ldots, p$, let $\mu_j^i(k) \in \mathbb{N}$ such that

$$\mu_j^i(k) \leq k, \tag{10}$$

and satisfying

$$\forall i, j \in \{1, \ldots, p\}, \quad \lim_{k \to +\infty} \mu_j^i(k) = +\infty. \tag{11}$$

An asynchronous iteration corresponding to f and starting with a given vector x^0 is defined recursively by

$$x_i^{k+1} = \begin{cases} x_i^k, & i \notin P^k, \\ f_i(x_1^{\mu_1^i(k)}, \ldots, x_p^{\mu_p^i(k)}), & i \in P^k. \end{cases} \tag{12}$$

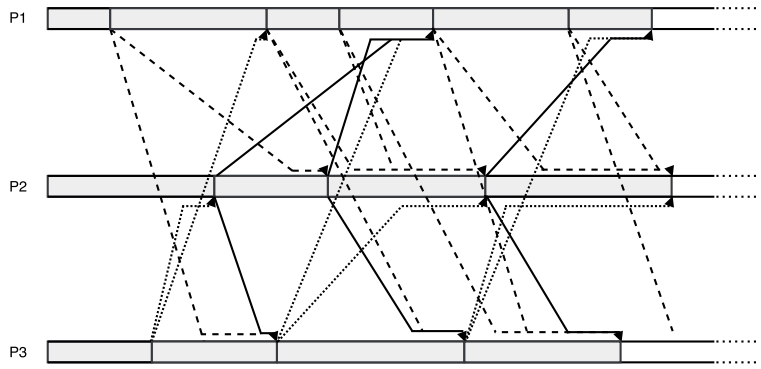

Figure 1: Example of the asynchronous iteration with asynchronous communication.

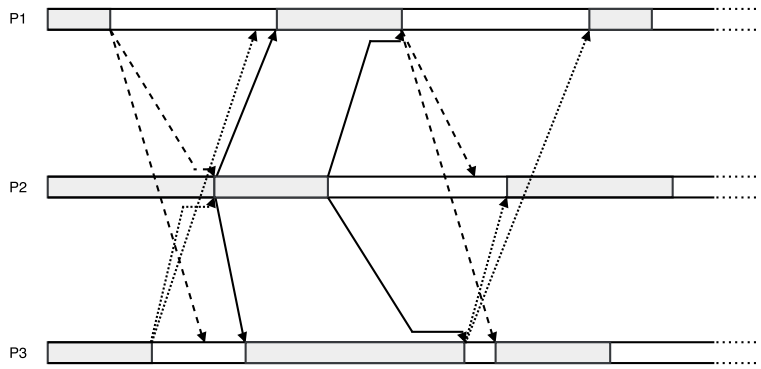

Figure 2: Example of the synchronous iteration with asynchronous communication.

Note that the assumption (9) illustrates that each processor will proceed the computation now and again, while (11) indicates that each component used by each processor will be updated eventually.

On the other hand, Algorithm 1 gives us a computational model of the basic asynchronous iteration.

Algorithm 1 Asynchronous iteration with asynchronous communication.

1: **while** not convergence **do**
2: Receive x from other processors
3: $x_i \leftarrow f_i(x)$
4: Send x_i to other processors
5: **end while**

It is important to notice that the asynchronous iteration functions well without the synchronization points, while the latter may become a decisive bottleneck. In other words, we proceed the next iteration using the latest local data rather than waiting for the newest information from other processors. Finally, we are interested in the operational process of such iteration. An typical example is showed in Figure 1. To make clear, we illustrate also the synchronous counterpart in Figure 2 whereby we can gain a better insight of the asynchronous iterative mechanism.

We now turn to the study of two-stage methods that can be applied to the parareal algorithm. The original asynchronous two-stage methods were introduced in [23], which are called outer asynchronous and totally asynchronous respectively. A well-known improvement consists of allowing the intermediate results from the inner level to be used by the other processors. As a consequence, this method may be performant in some cases taking

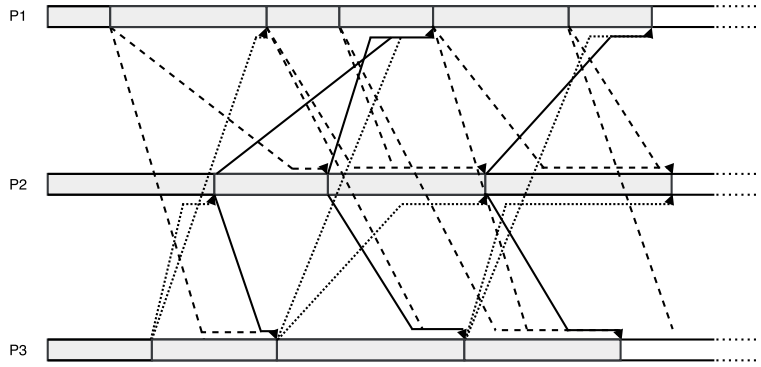

Figure 3: Example of the asynchronous two-stage iteration with flexible communication.

advantage of a better approximation to the solution.

It would obviously be possible to extent the model (12) to the two-stage situation. Let $f : E \times E \to E$ be the function defined by $f_i : E \times E \to E_i$. For $i = 1, \ldots, p$ and $j = 1, \ldots, p$, let $\rho_j^i(k) \in \mathbb{N}$ such that

$$\rho_j^i(k) \leq k, \tag{13}$$

and satisfying

$$\forall i, j \in \{1, \ldots, p\}, \quad \lim_{k \to +\infty} \rho_j^i(k) = +\infty. \tag{14}$$

We assume that the conditions (9), (10) and (11) are still vouched in such context. Then, an asynchronous two-stage iteration with flexible communication corresponding to f and starting with a given vector x^0 is defined recursively by

$$x_i^{k+1} = \begin{cases} x_i^k, & i \notin P^k, \\ f_i((x_1^{\mu_1^i(k)}, \ldots, x_p^{\mu_p^i(k)}), (x_1^{\rho_1^i(k)}, \ldots, x_p^{\rho_p^i(k)})), & i \in P^k. \end{cases} \tag{15}$$

Furthermore, we can establish the computational scheme from the aforementioned mathematical model, which is presented in Algorithm 2.

Algorithm 2 Asynchronous two-stage iteration with flexible communication.

```
1: while not convergence do
2:     Receive x from other processors
3:     x̂ ← x
4:     Send x_i to other processors
5:     while not precise enough do
6:         x_i ← f_i(x_i, x̂)
7:         Send x_i to other processors
8:     end while
9: end while
```

Clearly, the crucial property lies in the sending instruction of inner iteration whereby one can make a transmission without waiting for the local completion in such scope. In this context, we may fully exploit the latest local approximations to accelerate the global convergence. In the same manner, Figure 3 provides an example of the asynchronous two-stage iteration with flexible communication, which illustrates the context of inner/outer iteration (see, e.g., [19]).

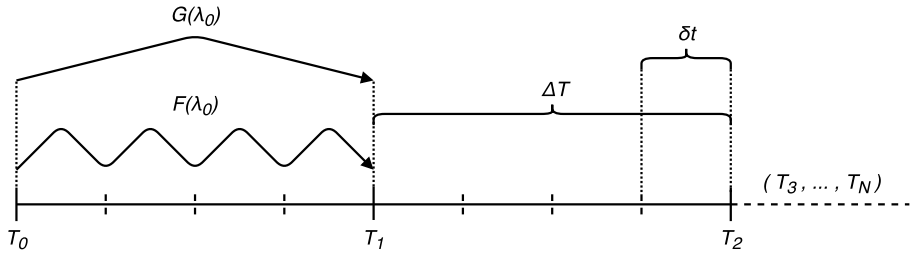

Figure 4: Parareal iterative scheme. F propagates step-by-step, whereas G performs only one step in each subdomain. We compute λ_{n+1} by combining the predictor and the corrector. Then, continue the next iteration.

3.2 Classical Parareal Algorithm

Given a second-order linear elliptic operator $\mathcal{L}$, consider the following time-dependent problem

$$\begin{cases} \frac{\partial u}{\partial t}(x,t) + \mathcal{L}u(x,t) = f(x,t), & t \in [0,T],\ x \in \Omega, \\ u(x,t) = u_0(x), & t = 0,\ x \in \Omega, \\ u(x,t) = g(x), & t \in [0,T],\ x \in \partial\Omega, \end{cases}$$

where the boundary $\partial\Omega$ is Lipschitz continuous. The above mathematical description can be decomposed into N sequential problems

$$0 = T_0 < \cdots < T_n = n\Delta T < \cdots < T_N = T.$$

By importing a function λ_n, we now reconstruct our problem as

$$\begin{cases} \frac{\partial u_n}{\partial t}(x,t) + \mathcal{L}u_n(x,t) = f_n(x,t), & t \in [T_n, T_{n+1}],\ x \in \Omega, \\ u_n(x,t) = \lambda_n(x), & t = T_n,\ x \in \Omega, \\ u_n(x,t) = g(x), & t \in [T_n, T_{n+1}],\ x \in \partial\Omega, \end{cases} \tag{16}$$

where $n = 0, \ldots, N-1$, together with the condition

$$\lambda_{n+1}(x) = \lim_{\epsilon \to 0} u_n(x, T_{n+1} - \epsilon).$$

Now we can solve the subproblems (16) independently with some essential message passing of the initial boundary values.

The parareal iterative scheme is driven by two operators, G and F, which are called coarse propagator and fine propagator respectively. On the other hand, we assume that the time-dependent problem is approximated by some appropriate classical discretization schemes. Let δt be a fine time step. Each subproblem will be solved at a time by G with respect to ΔT, and be further solved by F concerning δt. Note that the discretization schemes exploited for such two operators might be different. Finally, in order to approximate the solution of (16) in parallel, we arrive at an iterative method

$$\lambda_{n+1}^{k+1} = G(\lambda_n^{k+1}) + F(\lambda_n^k) - G(\lambda_n^k),$$

which is the parareal iterative scheme, where $G(\lambda_n^{k+1})$ is called predictor and the remain is called corrector, with $n = 0, \ldots, N-1$ and $\lambda_0^0 = u_0$. This is therefore a predictor-corrector scheme. Furthermore, we are obliged to respect two more conditions $\lambda_{n+1}^0 = G(\lambda_n^0)$ and $\lambda_0^{k+1} = \lambda_0^k$ within the iterations. To make clear, Figure 4 illustrates the aforementioned context and Algorithm 3 gives us the implementation of such method.

Algorithm 3 Classical parareal algorithm.

n : rank of current processor
$\lambda_0 = u_0$
for $i = 0$ to $n-1$ **do**
    $\lambda_0 = G(\lambda_0)$
end for
$w = G(\lambda_0)$
while not convergence **do**
    $v = F(\lambda_0)$
    wait for the update of λ_0 from processor $n-1$
    $\tilde{w} = G(\lambda_0)$
    $\lambda = \tilde{w} + v - w$
    send λ to processor $n+1$ as λ_0
    $w = \tilde{w}$
end while

3.3 Asynchronous Parareal Algorithm

Now we turn to the investigation of a modified parareal method based on asynchronous scheme. We choose equation (15) to establish the new model. Given $P^k \subseteq \{0, \ldots, N-1\}$ and $P^k \neq \emptyset$, consider the following iteration

$$\lambda_{n+1}^{k+1} = \begin{cases} \lambda_{n+1}^k, & n \notin P^k, \\ G(\lambda_n^{\mu_n(k)}) + F(\lambda_n^{\rho_n(k)}) - G(\lambda_n^{\rho_n(k)}), & n \in P^k, \end{cases}$$

where $\lambda_0^{k+1} = \lambda_0^k$, and satisfying

$$0 \leq \mu_j^i(k) \leq k+1, \quad 0 \leq \rho_j^i(k) \leq k,$$

and under the similar assumptions

$$\begin{cases} \mathrm{card}\{k \in \mathbb{N} \mid n \in P^k\} = +\infty, & n \in \{0, \ldots, N-1\}, \\ \lim\limits_{k \to +\infty} \mu_n(k) = \lim\limits_{k \to +\infty} \rho_n(k) = +\infty, & n \in \{0, \ldots, N-1\}, \end{cases}$$

We notice that the predictor and the corrector come from different iterations, so that we can treat parareal scheme as a special case of two-stage methods. It is seen that the classical parareal scheme will be obtained when setting $P^k = \{0, \ldots, N-1\}$, $\mu_n(k) = k+1$ and $\rho_n(k) = k$. As a consequence, the asynchronous parareal scheme is illustrated in Algorithm 4 whereby one can exploit for the implementation.

4 Implementation

4.1 Asynchronous Communication Library

The development of asynchronous communication library is interesting because they provide the reliable computational environments. However, the issues of termination and communication management are important, for which many theoretical investigations have been done, but few implementations are proposed. Several libraries have been proposed to deal with the problems (see, e.g., [6, 5, 4, 11, 14]), implemented upon Java or C++, but no one manages to build upon the MPI library, which is indeed widely used in the scientific domain.

Recently, JACK [39], an asynchronous communication kernel library, was proposed as the first MPI-based C++ library for parallel implementation of both classical and asynchronous iterations, which has been upgraded to the new version called JACK2. It offers a new high-level API, a simplified management of resources and several global convergence detectors.

Algorithm 4 Asynchronous parareal algorithm.

n : rank of current processor
$\lambda_0 = u_0$
for $i = 0$ to $n - 1$ **do**
    $\lambda_0 = G(\lambda_0)$
end for
$w = G(\lambda_0)$
while not convergence **do**
    $v = F(\lambda_0)$
    if detect λ_0 from processor $n - 1$ **then**
        update λ_0
    end if
    $\tilde{w} = G(\lambda_0)$
    $\lambda = \tilde{w} + v - w$
    send λ to processor $n + 1$ as λ_0
    $w = \tilde{w}$
end while

In the sequel, we will specify the implementation of the asynchronous parareal iterative scheme.

4.2 Preprocessing

We follow the primitive ideas of JACK2 that each element should be configured before processing including communication graph, communication buffers, computation residual and solution vectors. Hence, there are many but unambiguous works to be carried out in this cycle.

The parareal algorithm is a special case of domain decomposition methods, since each time frame only depends upon its predecessor, and essentially needed by its successor. We separate the neighbors into the outgoing links and the incoming links, and thus write the code as Listing 1. Notice that the first processor has no predecessor, whereas the last one has no successor.

Listing 1: Communication graph.

```
/* template <typename T, typename U> */
// T: float, double, ...
// U: int, long, ...
U numb_sneighb = 1;
U numb_rneighb = 1;
U* sneighb_rank = new U[1]; // outgoing links.
U* rneighb_rank = new U[1]; // incoming links.
```

The second component is the communication buffers, which is managed completely by the library, illustrated in Listing 2. Clearly, each processor needs to handle the information from one incoming neighbor and one outgoing neighbor, while each neighbor has an array of data. Therefore, buffers are constructed with the two-dimensional arrays.

Listing 2: Communication buffers.

```
/* template <typename T, typename U> */
U* sbuf_size = new U[1];
U* rbuf_size = new U[1];
sbuf_size[0] = numb_sub_domain;
rbuf_size[0] = numb_sub_domain;
T** send_buf = new T*[1]; // buffers for sending data.
T** recv_buf = new T*[1]; // buffers for receiving data.
```

```
send_buf[0] = new T[sbuf_size[0]];
recv_buf[0] = new T[rbuf_size[0]];
```

The computation residual involves an indication of norm, whereby we define the length of a vector, and thus measure the convergence results compared with a threshold. In Listing 3, as an example, we choose the L2-norm and present the configuration.

Listing 3: Computation residual.

```
/* template <typename T, typename U> */
T* res_vec_buf = new T[1]; // local residual vector.
U res_vec_size = 1;
T res_vec_norm; // norm of the global residual vector.
float norm_type = 2; // 2 for Euclidean norm, < 1 for maximum norm.
```

Finally, for the asynchronous parareal scheme, the solution vectors are the parameters to initialize the configuration of asynchronous iterations. Listing 4 illustrates the variables in demand.

Listing 4: Solution vectors.

```
/* template <typename T, typename U> */
T* sol_vec_buf; // local solution vector.
U sol_vec_size = numb_sub_domain;
int lconv_flag; // local convergence indicator.
```

In JACK2, we can see that *JACKComm* is a front-end interface to perform both blocking and nonblocking tasks. A simple way to initialize the communicator is shown in Listing 5.

Listing 5: Initialization of communicator.

```
// -- initializes MPI
MPI_Init(&argc, &argv);

JACKComm comm;
comm.Init(numb_sneighb, numb_rneighb, sneighb_rank, rneighb_rank, MPI_COMM_WORLD);
comm.Init(sbuf_size, rbuf_size, send_buf, recv_buf);
comm.Init(res_vec_size, res_vec_buf, &res_vec_norm, norm_type);
```

For the asynchronous mode, the library employs a member function in initializing the solution vectors, which can be easily overloaded for some advanced abilities.

Listing 6: Initialization of asynchronous mode.

```
if (async_flag) {
  comm.ConfigAsync(sol_vec_size, &sol_vec_buf, &lconv_flag, &recv_buf);
  comm.SwitchAsync();
}
```

We mention here that there are many classes behind the common front-end interface to handle the diverse problems, such as stopping criterion, norm computation and spanning tree construction. If necessary, moreover, one can switch to an appropriate convergence detection method within several choices equipped with some advanced configurations, which are still easy to manipulate. We do not pursue these features further and turn to the implementation of processing step.

4.3 Implementation of Parareal Algorithms

We implement the algorithms in two levels, which solve the sequential time-dependent problem and parallel-in-time problem respectively. It is seen that in parallel solver we solve

independently the sequential problems, leading to a composite pattern in our design. Our discussion focus on the parareal scheme applied to the partial differential equations. Hence, we give two classes, *PDESolver* and *Parareal*, to meet the needs. Moreover, for our specific problem, we need two instances to simulate coarse propagator and fine propagator, for which the initialization is illustrated as Listing 7.

Listing 7: Declaration of solvers.

```
/* template <typename T, typename U> */
PDESolver<T,U> coarse_pde;
PDESolver<T,U> fine_pde;
Vector<T,U> coarse_vec_U; // coarse results.
Vector<T,U> fine_vec_U; // fine results.
Vector<T,U> vec_U; // solution vector.
Vector<T,U> vec_U0; // initial vector.
```

There are still some initialization functions for the solvers which are unessential to be mentioned. The key part begins with some chores that must be handled before the main iteration. Following the aforementioned parareal algorithms, we illustrate such process, somewhat trivial, in Listing 8.

Listing 8: Initialization of iterations.

```
/* template <typename T, typename U> */
for (U i = 0; i < rank; i++) {
  coarse_pde.Integrate();
  vec_U0 = coarse_vec_U;
}
coarse_pde.Integrate();
vec_U = coarse_vec_U;
```

Now we have a dividing ridge. For the classical parareal algorithm, we follow Algorithm 3 and list the code as below. Note that the processor n will stop updating after $(n+1)$th iteration, which leads to a supplementary condition in the judging area to lighten the communication. Furthermore, we need to finalize the fine solution and wait for the global termination.

Listing 9: Synchronous parareal iterative process.

```
res_norm = res_thresh;
numb_iter = 0;
while (res_norm >= res_thresh && numb_iter < m_rank) {
  fine_pde.Integrate();
  comm.Recv();
  coarse_vec_U_prev = coarse_vec_U;
  coarse_pde.Integrate();
  vec_U_prev = vec_U;
  vec_U = coarse_vec_U + fine_vec_U - coarse_vec_U_prev;
  comm.Send();
  // -- |Un+1<k+1> - Un+1<k>|
  vec_local_res = vec_U - vec_U_prev;
  (*res_vec_buf) = vec_local_res.NormL2();
  comm.UpdateResidual();
  numb_iter++;
}
```

Listing 10: Synchronous parareal finalized process.

```
if (res_norm >= res_thresh) {
  fine_pde.Integrate();
  vec_U = fine_vec_U;
  comm.Send();
  // -- wait for global termination
```

```
  (*res_vec_buf) = 0.0;
  while (res_vec_norm >= res_thresh) {
    comm.UpdateResidual();
    numb_iter++;
  }
}
```

Finally, the asynchronous mode is more interesting for us, while the implementation is compact enough. Listing 11 illustrate a somewhat similar code as before.

Listing 11: Asynchronous parareal iterative process.

```
res_norm = res_thresh;
numb_iter = 0;
while (res_norm >= res_thresh) {
  fine_pde.Integrate();
  comm.Recv();
  coarse_vec_U_prev = coarse_vec_U;
  coarse_pde.Integrate();
  vec_U_prev = vec_U;
  vec_U = coarse_vec_U + fine_vec_U - coarse_vec_U_prev;
  comm.Send();
  // -- |Un+1<k+1> - Un+1<k>|
  vec_local_res = vec_U - vec_U_prev;
  (*res_vec_buf) = vec_local_res.NormL2();
  lconv_flag = ((*res_vec_buf) < res_thresh);
  comm.UpdateResidual();
  numb_iter++;
}
```

Notice that function *Integrate* gives the same effect as $G(\lambda_n)$ or $F(\lambda_n)$. In the code above, we use *vec_ U0* in each processor as incoming information and employ the two propagator to obtain the intermediate data, whereas *vec_ U*, the solution vector, is computed eventually by the predictor-corrector scheme. *lconv_flag* is a convergence indicator equipped by the library, whereby function *UpdateResidual* can invoke other objects to communicate the residual information. It is seen that the programmer should guarantee the sending and receiving buffers corresponding to the *vec_ U* and *vec_ U0*. In practice, such behavior depends on the mathematical library used in the project.

5 Numerical Experiments

In this section, we are interested in the numerical performance of asynchronous parareal method. We utilize Alinea [34] to carry out the mathematical operations, which is implemented in C++ for both central processing unit and graphic processing unit devices. Note that it has basic linear algebra operations [1] and plenty of linear system solvers [36, 2, 35], together with some energy consumption optimization [38] and spatial domain decomposition methods [37]. Besides, the experiments are executed on the SGI ICE X clusters connected with InfiniBand. Each node includes two Intel Xeon E5-2670 v3 2.30 GHz CPUs. Finally, SGI Message Passing Toolkit 2.14 provides the MPI environment.

As we mentioned above, the application is European option pricing problem, and we employ the Black-Scholes model to predict the target price. The basic idea is, obviously, that we can use heat equation to simplify the computation. After deciding the appropriate discretization schemes, we need to solve the remain equation in the time domain. In the sequel, we assume that both coarse problem and fine problem are approximated by the implicit Euler method, while testing different high-order schemes are beyond the purpose of this paper. Moreover, let us fix volatility and risk-free interest rate as $\sigma = 0.2$ and $r = 0.05$, which has few influence to the parareal algorithms. Finally, we choose finite difference method to discretize the spatial domain, thereby a variable m representing the number of sub-intervals has to be determined.

We first illustrate some results which means to prove the precision of asynchronous parareal scheme. In Table 1, we vary ΔT to simulate different time to maturity, and further obtain different option prices, where we can see that the method is precise enough.

Table 1: Asynchronous parareal results with approximate option prices V_a, exact option prices V_e, absolute error ϵ_a, relative error ϵ_r and number of iterations I, given $N = 20$, $m = 250$, $\delta t = 0.001$, $S = 50$, $E = 60$.

ΔT	V_a	V_e	ϵ_a	ϵ_r	$I_{\min}$	$I_{\max}$	I_{mean}	Time
0.05	1.6297	1.6237	0.0060	0.0037	33	43	37	0.600
0.20	8.3064	8.3022	0.0042	0.0005	34	47	39	2.453
0.35	13.9586	13.9541	0.0045	0.0003	33	43	37	4.039
0.50	18.7968	18.7918	0.0050	0.0003	34	47	40	6.123
0.65	22.9713	22.9659	0.0054	0.0002	34	45	39	7.769
0.80	26.5845	26.5796	0.0049	0.0002	38	47	42	10.337
0.95	29.7150	29.7125	0.0025	0.0001	35	45	39	11.459

In the case of convergence properties, we illustrate an example In Figure 5, over which we can see, tidily enough, that each processor converges fast.

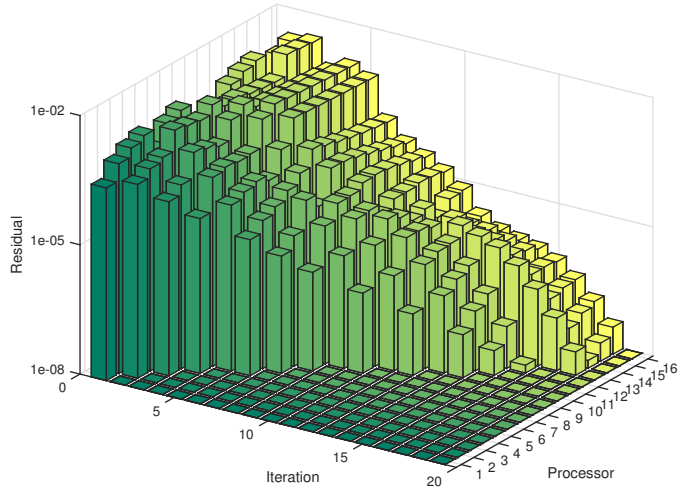

Figure 5: Asynchronous parareal iterations for 16 cores. Unlike other asynchronous methods, parareal scheme performs a tidy convergence process.

For the synchronous counterpart, as mentioned before, the processor n will stop updating after $(n+1)$th iteration. We notice that the asynchronous parareal are similar to that case, but bring out different number of iterations when keeping a better view.

We now consider the running time. The first case leads to a fixed time to maturity, while changing the number of processors dramatically, shown in Figure 6.

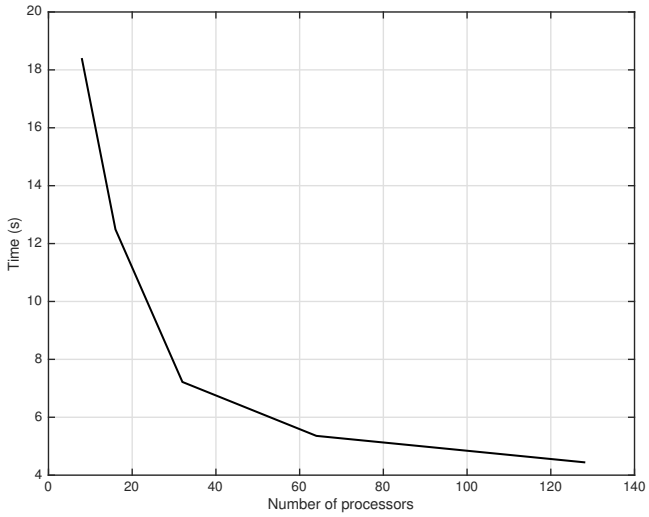

Figure 6: Asynchronous parareal iterations for fixed time to maturity.

We vary ΔT with respect to N to keep the problem immutable with some given parameters. Finally, we compare the performance of asynchronous parareal with its synchronous counterpart, which leads to Table 2.

Table 2: Comparison of Synchronous and Asynchronous Parareal Scheme, given $m = 150$, $S = 25$, $E = 30$, $\delta t = 0.001$, $\Delta T = 0.1$.

	Synchronous		Asynchronous			
N	Iter.	Time	$I_{\min}$	$I_{\max}$	I_{mean}	Time
16	11	0.620	22	30	26	0.490
32	11	0.781	30	47	40	0.677
64	11	0.971	44	77	60	0.947

The results illustrate that the asynchronous scheme requires more iterations with the increase of processors, whereas the number of iterations of the synchronous version remains the same throughout the test. We notice that the asynchronous version is faster. Let us mention here that the environment is not distributed, thereby the scene of large communication delays can hardly appear.

6 Conclusions

In this paper we investigated a modified parareal method with respect to the asynchronous iterative scheme. We first proposed a brand-new scheme from the traditional parallel theories, upon which an attractive model was established. Note that we applied asynchronous iterations to the predictor-corrector scheme, instead of the classical inner-outer scheme, thereby we can make use of the two-stage model to derive our target equations. To overcome the difficulty of implementation, an asynchronous communication library has been adopted to facilitate our programming. We illustrated the design philosophy in detail and gave several numerical results, from which, as expected, we illustrated the excellent precision and the conditional efficiency for the target approach. Notice that the Black-Scholes equation is too simple to accurately predict the real financial market, thereby we could proceed our research with a somewhat challenging situation, such as a mutable volatility or other types of option

pricing problem. More important, there are still many theoretical research could be done, upon which a comprehensive analysis is under investigation by the authors.

References

[1] A.-K. C. Ahamed and F. Magoulès. Fast sparse matrix-vector multiplication on graphics processing unit for finite element analysis. In *14th IEEE Int. Conf. on High Performance Computing and Communications, Liverpool, UK, June 25-27, 2012*. IEEE, 2012.

[2] A.-K. C. Ahamed and F. Magoulès. Iterative methods for sparse linear systems on graphics processing unit. In *14th IEEE Int. Conf. on High Performance Computing and Communications, Liverpool, UK, June 25-27, 2012*. IEEE, 2012.

[3] J. M. Bahi, S. Contassot-Vivier, and R. Couturier. *Parallel Iterative Algorithms: From Sequential to Grid Computing*. CRC Press, 1 edition, 2007.

[4] J. M. Bahi, R. Couturier, and P. Vuillemin. JaceP2P: an environment for asynchronous computations on peer-to-peer networks. In *2006 IEEE International Conference on Cluster Computing*, pages 1–10, 2006.

[5] J. M. Bahi, R. Couturier, and P. Vuillemin. JaceV: A programming and execution environment for asynchronous iterative computations on volatile nodes. In *2006 International Conference on High Performance Computing for Computational Science*, pages 79–92. Springer Berlin Heidelberg, 2006.

[6] J. M. Bahi, S. Domas, and K. Mazouzi. Jace: a Java environment for distributed asynchronous iterative computations. In *12th Euromicro Conference on Parallel, Distributed and Network-Based Processing*, pages 350–357, 2004.

[7] G. M. Baudet. Asynchronous iterative methods for multiprocessors. *J. ACM*, 25(2):226–244, 1978.

[8] D. P. Bertsekas. Distributed asynchronous computation of fixed points. *Mathematical Programming*, 27(1):107–120, 1983.

[9] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[10] F. Black and M. Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81(3):637–654, 1973.

[11] J.-C. Charr, R. Couturier, and D. Laiymani. JACEP2P-V2: A fully decentralized and fault tolerant environment for executing parallel iterative asynchronous applications on volatile distributed architectures. In *2009 International Conference on Advances in Grid and Pervasive Computing*, pages 446–458. Springer Berlin Heidelberg, 2009.

[12] P. Chartier and B. Philippe. A parallel shooting technique for solving dissipative ODE's. *Computing*, 51(3):209–236, 1993.

[13] D. Chazan and W. Miranker. Chaotic relaxation. *Linear Algebra and its Applications*, 2(2):199–222, 1969.

[14] R. Couturier and S. Domas. CRAC: a grid environment to solve scientific applications with asynchronous iterative algorithms. In *2007 IEEE International Parallel and Distributed Processing Symposium*, pages 1–8, 2007.

[15] J. C. Cox and S. A. Ross. The valuation of options for alternative stochastic processes. *Journal of Financial Economics*, 3(1):145–166, 1976.

[16] J. C. Cox, S. A. Ross, and M. Rubinstein. Option pricing: A simplified approach. *Journal of Financial Economics*, 7(3):229–263, 1979.

[17] P. Deuflhard. *Newton Methods for Nonlinear Problems: Affine Invariance and Adaptive Algorithms*, volume 35 of *Springer Series in Computational Mathematics*. Springer-Verlag Berlin Heidelberg, 2004.

[18] J.-C. Duan. The GARCH option pricing model. *Mathematical Finance*, 5(1):13–32, 1995.

[19] D. El Baz, A. Frommer, and P. Spiteri. Asynchronous iterations with flexible communication: Contracting operators. *J. Comput. Appl. Math.*, 176(1):91–103, 2005.

[20] D. El Baz, P. Spitéri, J. C. Miellou, and D. Gazen. Asynchronous iterative algorithms with flexible communication for nonlinear network flow problems. *Journal of Parallel and Distributed Computing*, 38(1):1–15, 1996.

[21] M. N. El Tarazi. Some convergence results for asynchronous algorithms. *Numerische Mathematik*, 39(3):325–340, 1982.

[22] C. Farhat and M. Chandesris. Time-decomposed parallel time-integrators: Theory and feasibility studies for fluid, structure, and fluid–structure applications. *International Journal for Numerical Methods in Engineering*, 58(9):1397–1434, 2003.

[23] A. Frommer and D. B. Szyld. Asynchronous two-stage iterative methods. *Numerische Mathematik*, 69(2):141–153, 1994.

[24] A. Frommer and D. B. Szyld. Asynchronous iterations with flexible communication for linear systems. *Calculateurs Parallèles*, 10:91–103, 1998.

[25] A. Frommer and D. B. Szyld. On asynchronous iterations. *Journal of Computational and Applied Mathematics*, 123(1–2):201–216, 2000.

[26] M. J. Gander and L. Halpern. Méthodes de relaxation d'ondes (SWR) pour l'équation de la chaleur en dimension 1. *Comptes Rendus Mathematique*, 336(6):519–524, 2003.

[27] M. J. Gander and H. Zhao. Overlapping Schwarz waveform relaxation for the heat equation in n dimensions. *BIT Numerical Mathematics*, 42(4):779–795, 2002.

[28] J. M. Harrison and D. M. Kreps. Martingales and arbitrage in multiperiod securities markets. *Journal of Economic Theory*, 20(3):381–408, 1979.

[29] G. Horton and V. Stefan. A space-time multigrid method for parabolic partial differential equations. *SIAM Journal on Scientific Computing*, 16(4):848–864, 1995.

[30] J. Hull and A. White. The pricing of options on assets with stochastic volatilities. *The Journal of Finance*, 42(2):281–300, 1987.

[31] K. Itô. *On Stochastic Differential Equations*. Memoris of the American Mathematical Society, 1951.

[32] B. Leimkuhler. Timestep acceleration of waveform relaxation. *SIAM Journal on Numerical Analysis*, 35(1):31–50, 1998.

[33] J.-L. Lions, Y. Maday, and G. Turinici. Résolution d'EDP par un schéma en temps "pararéel". *Comptes rendus de l'Académie des sciences - Série I - Mathématique*, 332(7):661–668, 2001.

[34] F. Magoulès and A.-K. C. Ahamed. Alinea: An advanced linear algebra library for massively parallel computations on graphics processing units. *The International Journal of High Performance Computing Applications*, 29(3):284–310, 2015.

[35] F. Magoulès, A.-K. C. Ahamed, and R. Putanowicz. Auto-tuned Krylov methods on cluster of graphics processing unit. *International Journal of Computer Mathematics*, 92(6):1222–1250, 2015.

[36] F. Magoulès, A.-K. C. Ahamed, and R. Putanowicz. Fast iterative solvers for large compressed-sparse row linear systems on graphics processing unit. *Pollack Periodica*, 10(1):3–18, 2015.

[37] F. Magoulès, A.-K. C. Ahamed, and R. Putanowicz. Optimized Schwarz method without overlap for the gravitational potential equation on cluster of graphics processing unit. *International Journal of Computer Mathematics*, 93(6):955–980, 2016.

[38] F. Magoulès, A.-K. C. Ahamed, and A. Suzuki. Green computing on graphics processing units. *Concurrency and Computation: Practice and Experience*, 28(16):4305–4325, 2016.

[39] F. Magoulès and G. Gbikpi-Benissan. JACK: an asynchronous communication kernel library for iterative algorithms. *The Journal of Supercomputing*, pages 1–20, 2017.

[40] F. Magoulès, D. B. Szyld, and C. Venet. Asynchronous optimized Schwarz methods with and without overlap. *Numerische Mathematik*, 137:199–227, 2017.

[41] F. Magoulès and C. Venet. Asynchronous iterative sub-structuring methods. *Mathematics and Computers in Simulation*, submitted for publication.

[42] R. C. Merton. Theory of rational option pricing. *Bell Journal of Economics*, 4(1):141–183, 1973.

[43] R. C. Merton. On the pricing of corporate debt: the risk structure of interest rates. *The Journal of Finance*, 29(2):449–470, 1974.

[44] J.-C. Miellou. Algorithmes de relaxation chaotique à retards. *ESAIM: Mathematical Modelling and Numerical Analysis*, 9(R1):55–82, 1975.

[45] J.-C. Miellou, D. El Baz, and P. Spitéri. A new class of asynchronous iterative algorithms with order intervals. *AMS Mathematics of Computation*, 67(221):237–255, 1998.

[46] J.-C. Miellou and P. Spitéri. Un critère de convergence pour des méthodes générales de point fixe. *ESAIM: Mathematical Modelling and Numerical Analysis*, 19(4):645–669, 1985.