Simulation of Asynchronous Iterative Algorithms Using SimGrid

Charles Emile Ramamonjisoa*, Lilia Ziane Khodja†, David Laiymani*, Arnaud Giersch* and Raphaël Couturier*
*Femto-ST Institute – DISC Department
Université de Franche-Comté, IUT de Belfort-Montbéliard
19 avenue du Maréchal Juin, BP 527, 90016 Belfort cedex, France
Email: {charles.ramamonjisoa,david.laiymani,arnaud.giersch,raphael.couturier}@univ-fcomte.fr
†Inria Bordeaux Sud-Ouest
200 avenue de la Vieille Tour, 33405 Talence cedex, France
Email: lilia.ziane@inria.fr

***Abstract*—Synchronous iterative algorithms are often less scalable than asynchronous iterative ones. Performing large scale experiments with different kind of network parameters is not easy because with supercomputers such parameters are fixed. So, one solution consists in using simulations first in order to analyze what parameters could influence or not the behavior of an algorithm. In this paper, we show that it is interesting to use SimGrid to simulate the behavior of asynchronous iterative algorithms. For that, we compare the behavior of a synchronous GMRES algorithm with an asynchronous multisplitting one with simulations which let us easily choose some parameters. Both codes are real MPI codes and simulations allow us to see when the asynchronous multisplitting algorithm can be more efficient than the GMRES one to solve a 3D Poisson problem.**

I. Introduction

Parallel computing and high performance computing (HPC) are becoming more and more imperative to solve various problems raised by researchers on various scientific disciplines but also by industrialists in the field. Indeed, the increasing complexity of these requested applications combined with a continuous increase of their sizes lead to write distributed and parallel algorithms requiring significant hardware resources (grid computing, clusters, broadband network, etc.) but also a non-negligible CPU execution time. We consider in this paper a class of highly efficient parallel algorithms called *iterative algorithms* executed in a distributed environment. As their name suggests, these algorithms solve a given problem by successive iterations ($X_{n+1} = f(X_n)$) from an initial value X_0 to find an approximate value X^* of the solution with a very low residual error. Several well-known methods demonstrate the convergence of these algorithms [1], [2].

Parallelization of such algorithms generally involves the division of the problem into several *blocks* that will be solved in parallel on multiple processing units. The latter will communicate each intermediate results before a new iteration starts and until the approximate solution is reached. These parallel computations can be performed either in a *synchronous* mode, where a new iteration begins only when all nodes communications are completed, or in an *asynchronous* mode where processors can continue independently with no synchronization points [3]. In this case, local computations do not need to wait for required data. Processors can then perform their iterations with the data present at that time. Even if the number of required iterations before the convergence is generally greater than in the synchronous case, asynchronous iterative algorithms can significantly reduce overall execution times by suppressing idle times due to synchronizations especially in a grid computing context (see [2] for more details).

Parallel applications based on a synchronous or asynchronous iteration model may have different configuration and deployment requirements. Quantifying their resource allocation policies and application scheduling algorithms in grid computing environments under varying load, CPU power and network speeds are very costly, very labor intensive and very time consuming [4]. The case of asynchronous iterative algorithms is even more problematic since they are very sensitive to the execution environment context. For instance, variations in the network bandwidth (intra and inter-clusters), in the number and the power of nodes, in the number of clusters... can lead to very different number of iterations and so to very different execution times. Then, it appears that the use of simulation tools to explore various platform scenarios and to run large numbers of experiments quickly can be very promising.

Thus, using a simulation environment to execute parallel iterative algorithms can prove to be very interesting to reduce the highly cost of access to computing resources: (1) for the applications development life cycle and in code debugging (2) and in production to get results in a reasonable execution time with a simulated infrastructure not accessible with physical resources. Indeed, to find optimal configurations giving the best results with a lowest residual error and in the best execution time is very challenging for large scale distributed iterative asynchronous algorithms

To our knowledge, there is no existing work on the large-scale simulation of a real asynchronous iterative application. **The contribution of the present paper can be summarized in two main points**. First we give a first approach of the simulation of asynchronous iterative algorithms using a simulation tool (i.e. the SimGrid toolkit [5]). Second, we confirm the efficiency of the asynchronous multisplitting algorithm by comparing its performances with the synchronous GMRES (Generalized Minimal Residual) method [6]. Both these codes can be used to solve large linear systems. In this paper, we focus on a 3D Poisson problem. We show that, with minor modifications of the initial MPI code, the SimGrid toolkit allows us to perform a test campaign of a real asynchronous iterative application on different computing architectures. SimGrid has allowed us to launch the application from a modest computing infrastructure by simulating different distributed architectures composed by clusters nodes interconnected by variable speed networks. Parameters of the network platforms are the bandwidth and the latency of inter cluster network. Parameters on the cluster's architecture are the number of machines and the computation power of a machine. Simulations show that the asynchronous multisplitting algorithm can solve the 3D Poisson problem approximately twice faster than GMRES with two distant clusters. In this way, we present an original solution to optimize the use of a simulation tool to run efficiently an asynchronous iterative parallel algorithm in a grid architecture

This article is structured as follows: after this introduction, the next section will give a brief description of the iterative asynchronous model. Then, the simulation framework SimGrid is presented with the settings to create various distributed architectures. Then, the multisplitting method is presented, it is based on GMRES to solve each block obtained from the splitting. This code is written with MPI primitives and its adaptation to SimGrid with SMPI (Simulated MPI) is detailed in the next section. At last, the simulation results carried out will be presented before some concluding remarks and future works.

II. Motivations and scientific context

As described in the introduction, parallel iterative methods are now widely used in many scientific domains. They can be classified in three main classes depending on how iterations and communications are managed (for more details readers can refer to [3]). In the synchronous iterations model, data are exchanged at the end of each iteration. All the processors must begin the same iteration at the same time and important idle times on processors are generated. It is possible to use asynchronous communications, in this case, the model can be compared to the previous one except that data required on another processor are sent asynchronously i.e. without stopping current computations. This technique allows communications to be partially overlapped by computations but unfortunately, the overlapping is only partial and important idle times remain. It is clear that, in a grid computing context, where the number of computational nodes is large, heterogeneous and widely distributed, the idle times generated by synchronizations are very penalizing. One way to overcome this problem is to use the asynchronous iterations model. Here, local computations do not need to wait for required data. Processors can then perform their iterations with the data present at that time. Figure 1 illustrates this model where the gray blocks represent the computation phases. With this algorithmic model, the number of iterations required before the convergence is generally greater than for the two former classes. But, and as detailed in [3], asynchronous iterative algorithms can significantly reduce overall execution times by suppressing idle times due to synchronizations especially in a grid computing context.

In the context of asynchronous algorithms, the number of iterations to reach the convergence depends on the delay of the messages. With synchronous iterations, the number of iterations is exactly the same than in the sequential mode (if

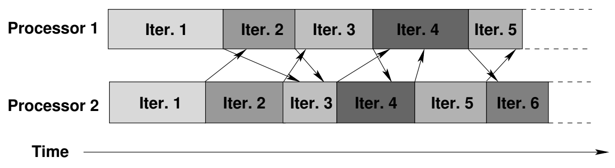

Figure 1. The asynchronous iterations model

the parallelization process does not change the algorithm). So the difficulty with asynchronous iterative algorithms comes from the fact that it is necessary to run the algorithm with real data. Indeed, from one execution to the other the order of messages will change and the number of iterations to reach the convergence will also change. According to all the parameters of the platform (number of nodes, power of nodes, inter and intra clusters bandwidth and latency, etc.) and of the algorithm (number of splittings with the multisplitting algorithm), the multisplitting code will obtain the solution more or less quickly. Of course, the GMRES method also depends on the same parameters. As it is difficult to have access to many clusters, grids or supercomputers with many different network parameters, it is interesting to be able to simulate the behavior of asynchronous iterative algorithms before being able to run real experiments.

III. SimGrid

SimGrid [5], [7], [8] is a simulation framework to study the behavior of large-scale distributed systems. As its name suggests, it emanates from the grid computing community, but is nowadays used to study grids, clouds, HPC or peer-to-peer systems. The early versions of SimGrid date back from 1999, but it is still actively developed and distributed as an open source software. Today, it is one of the major generic tools in the field of simulation for large-scale distributed systems.

SimGrid provides several programming interfaces: MSG to simulate Concurrent Sequential Processes, SimDAG to simulate DAGs of (parallel) tasks, and SMPI to run real applications written in MPI [9]. Apart from the native C interface, SimGrid provides bindings for the C++, Java, Lua and Ruby programming languages. SMPI is the interface that has been used for the work described in this paper. The SMPI interface implements about 80 % of the MPI 2.0 standard [10], and supports applications written in C or Fortran, with little or no modifications.

Within SimGrid, the execution of a distributed application is simulated by a single process. The application code is really executed, but some operations, like communications, are intercepted, and their running time is computed according to the characteristics of the simulated execution platform. The description of this target platform is given as an input for the execution, by means of an XML file. It describes the properties of the platform, such as the computing nodes with their computing power, the interconnection links with their bandwidth and latency, and the routing strategy. The scheduling of the simulated processes, as well as the simulated running time of the application are computed according to these properties.

To compute the durations of the operations in the simulated world, and to take into account resource sharing (e.g. bandwidth sharing between competing communications), SimGrid uses a fluid model. This allows users to run relatively fast simulations, while still keeping accurate results [10], [11]. Moreover, depending on the simulated application, SimGrid/SMPI allows to skip long lasting computations and to only take their duration into account. When the real computations cannot be skipped, but the results are unimportant for the simulation results, it is also possible to share dynamically allocated data structures between several simulated processes, and thus to reduce the whole memory consumption. These two techniques can help to run simulations on a very large scale.

The validity of simulations with SimGrid has been asserted by several studies. See, for example, [11] and articles referenced therein for the validity of the network models. Comparisons between real execution of MPI applications on the one hand, and their simulation with SMPI on the other hand, are presented in [12], [13], [10]. All these works conclude that SimGrid is able to simulate pretty accurately the real behavior of the applications.

IV. Simulation of the multisplitting method

A. The multisplitting method

Let $Ax = b$ be a large sparse system of n linear equations in $\mathbb{R}$, where A is a sparse square and nonsingular matrix, x is the solution vector and b is the right-hand side vector. We use a multisplitting method based on the block Jacobi splitting to solve this linear system on a large scale platform composed of L clusters of processors [14]. In this case, we apply a row-by-row splitting without overlapping

$$\left(\begin{array}{ccc} A_{11} & \cdots & A_{1L} \\ \vdots & \ddots & \vdots \\ A_{L1} & \cdots & A_{LL} \end{array}\right) \times \left(\begin{array}{c} X_1 \\ \vdots \\ X_L \end{array}\right) = \left(\begin{array}{c} B_1 \\ \vdots \\ B_L \end{array}\right)$$

in such a way that successive rows of matrix A and both vectors x and b are assigned to one cluster, where for all $\ell, m \in \{1, \ldots, L\}$, $A_{\ell m}$ is a rectangular block of A of size $n_\ell \times n_m$, X_ℓ and B_ℓ are sub-vectors of x and b, respectively, of size n_ℓ each, and $\sum_\ell n_\ell = \sum_m n_m = n$.

The multisplitting method proceeds by iteration to solve in parallel the linear system on L clusters of processors, in such a way each sub-system

$$\left\{ \begin{array}{l} A_{\ell\ell} X_\ell = Y_\ell, \text{ such that} \\ Y_\ell = B_\ell - \displaystyle\sum_{\substack{m=1 \\ m \neq \ell}}^{L} A_{\ell m} X_m \end{array} \right. \quad (1)$$

is solved independently by a cluster and communications are required to update the right-hand side sub-vector Y_ℓ, such that the sub-vectors X_m represent the data dependencies between the clusters. As each sub-system (1) is solved in parallel by a cluster of processors, our multisplitting method uses an iterative method as an inner solver which is easier to parallelize and more scalable than a direct method. In this work, we use the parallel algorithm of GMRES method [6] which is one of the most used iterative method by many researchers.

The algorithm in Figure 2 shows the main key points of the multisplitting method to solve a large sparse linear system. This algorithm is based on an outer-inner iteration

```
Input: $A_\ell$ (sparse sub-matrix), $B_\ell$ (right-hand side sub-vector)
Output: $X_\ell$ (solution sub-vector)

1:  Load $A_\ell$, $B_\ell$
2:  Set the initial guess $x^0$
3:  for $k = 0, 1, 2, \ldots$ until the global convergence do
4:      Restart outer iteration with $x^0 = x^k$
5:      Inner iteration: InnerSolver($x^0$, $k + 1$)
6:      Send shared elements of $X_\ell^{k+1}$ to neighboring clusters
7:      Receive shared elements in $\{X_m^{k+1}\}_{m \neq \ell}$
8:  end for

9:  function InnerSolver($x^0$, $k$)
10:     Compute local right-hand side $Y_\ell$:
            $Y_\ell = B_\ell - \sum_{m=1, m\neq\ell}^{L} A_{\ell m} X_m^0$
11:     Solving sub-system $A_{\ell\ell} X_\ell^k = Y_\ell$ with the parallel
        GMRES method
12:     return $X_\ell^k$
13: end function
```

Figure 2. A multisplitting solver with GMRES method

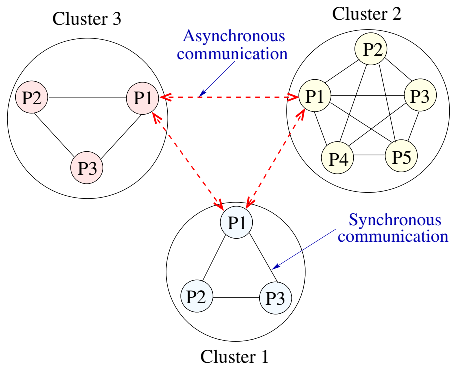

Figure 3. Example of three distant clusters of processors.

method where the parallel synchronous GMRES method is used to solve the inner iteration. It is executed in parallel by each cluster of processors. For all $\ell, m \in \{1, \ldots, L\}$, the matrices and vectors with the subscript ℓ represent the local data for cluster ℓ, while $\{A_{\ell m}\}_{m \neq \ell}$ are off-diagonal matrices of sparse matrix A and $\{X_m\}_{m \neq \ell}$ contain vector elements of solution x shared with neighboring clusters. At every outer iteration k, asynchronous communications are performed between processors of the local cluster and those of distant clusters (lines 6 and 7 in Figure 2). The shared vector elements of the solution x are exchanged by message passing using MPI non-blocking communication routines.

The global convergence of the asynchronous multisplitting solver is detected when the clusters of processors have all converged locally. We implemented the global convergence detection process as follows. On each cluster a master processor is designated (for example the processor with rank 1) and masters of all clusters are interconnected by a virtual unidirectional ring network (see Figure 3). During the resolution, a Boolean token circulates around the virtual ring from a master processor to another until the global convergence is achieved. So, starting from the cluster with rank 1, each master processor ℓ sets the token to *True* if the local convergence is achieved or to *False* otherwise, and sends it to master processor $\ell+1$. Finally, the global convergence is detected when the master of cluster 1 receives from the master of cluster L a token set to *True*. In this case, the master of cluster 1 broadcasts a stop message to the masters of other clusters. In this work, the local convergence on each cluster ℓ is detected when the following condition is satisfied

$$(k = MaxIter) \text{ or } (\|X_\ell^k - X_\ell^{k+1}\|_\infty \le \epsilon)$$

where $MaxIter$ is the maximum number of outer iterations and ϵ is the tolerance threshold of the error computed between two successive local solution X_ℓ^k and X_ℓ^{k+1}. It should be noted that with asynchronous iterative algorithms, we cannot use a classical norm (which would require to synchronize all processors), such as $\|X_\ell^k - X_\ell^{k+1}\|_2$ for example. Nevertheless, in our experiments, we check that the final result is correct, for this we compute the precision with $max_i|A*x-b|_i$.

In this paper, we solve the 3D Poisson problem whose mathematical model is

$$\left\{\begin{array}{l} \nabla^2 u = f \text{ in } \Omega \\ u = 0 \text{ on } \Gamma = \partial\Omega \end{array}\right. \tag{2}$$

where ∇^2 is the Laplace operator, f and u are real-valued functions, and $\Omega = [0,1]^3$. The spatial discretization with a finite differences scheme reduces problem (2) to a system of sparse linear equations. Our multisplitting method solves the 3D Poisson problem using a seven point stencil whose general expression could be written as

$$\begin{array}{l} u(x-1,y,z) + u(x,y-1,z) + u(x,y,z-1) \\ +u(x+1,y,z) + u(x,y+1,z) + u(x,y,z+1) \\ -6u(x,y,z) = h^2 f(x,y,z), \end{array} \tag{3}$$

where h is the distance between two adjacent elements in the spatial discretization scheme and the iteration matrix A of size $N_x \times N_y \times N_z$ of the discretized linear system is sparse, symmetric and positive definite.

The parallel solving of the 3D Poisson problem with our multisplitting method requires a data partitioning of the problem between clusters and between processors within a cluster. We have chosen the 3D partitioning instead of the row-by-row partitioning one in order to reduce the data exchanges at sub-domain boundaries. Figure 4 shows an example of the data partitioning of the 3D Poisson problem between two clusters of processors, where each sub-problem is assigned to a processor. In this context, a processor has at most six neighbors within a cluster or in distant clusters with which it shares data at sub-domain boundaries.

B. Simulation of the multisplitting method using SimGrid and SMPI

We did not encounter major blocking problems when adapting the multisplitting algorithm previously described to a simulation environment like SimGrid unless some code debugging. Indeed, apart from the review of the program sequence for asynchronous exchanges between processors within a cluster or between clusters, the algorithm was executed successfully with SMPI and provided identical outputs as those obtained with direct execution under MPI. For the synchronous GMRES method, the execution of the program raised no particular issue but in the asynchronous multisplitting method, the review of the sequence of `MPI_Isend`, `MPI_Irecv` and `MPI_Waitall` instructions and with the addition of the primitive `MPI_Test` was needed to avoid a memory fault due to an infinite loop resulting from the non-convergence of the algorithm. Note here that the use of SMPI functions optimizer for memory footprint and CPU usage is not recommended knowing that one wants to get real results by simulation. As mentioned, upon this adaptation, the algorithm is executed as in real life in the simulated environment after the following minor changes. The scope of all declared global variables have been moved to local subroutines. Indeed, global variables generate side effects arising from the concurrent access of shared memory used by threads simulating each computing unit in the SimGrid architecture. In total, the initial MPI program running on the simulation environment SMPI gave after a very simple adaptation the same results as those obtained in a real environment. We have successfully executed the code for the synchronous GMRES algorithm compared with our asynchronous multisplitting algorithm after few modifications.

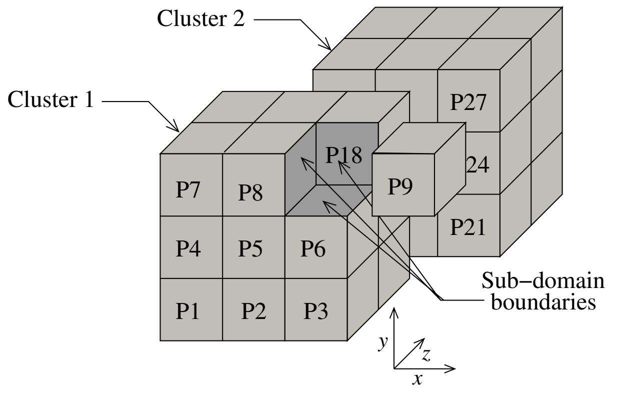

Figure 4. Example of the 3D data partitioning between two clusters of processors.

V. Simulation results

When the *real* application runs in the simulation environment and produces the expected results, varying the input parameters and the program arguments allows us to compare outputs from the code execution. We have noticed from this study that the results depend on the following parameters:

- At the network level, we found that the most critical values are the bandwidth and the network latency.
- Host processor power (GFlops) can also influence the results.
- Finally, when submitting job batches for execution, the arguments values passed to the program like the maximum number of iterations or the precision are critical. They allow us to ensure not only the convergence of the algorithm but also to get the main objective in getting an execution time with the asynchronous multisplitting less than with synchronous GMRES.

The ratio between the simulated execution time of synchronous GMRES algorithm compared to the asynchronous multisplitting algorithm ($t_{\text{GMRES}}/t_{\text{Multisplitting}}$) is defined as the

Table I. Relative gain of the multisplitting algorithm compared to GMRES for different configurations with 2 clusters, each one composed of 50 nodes. Latency = 20ms

bandwidth (Mbit/s)	5	5	5	5	5
power (GFlops)	1	1	1	1.5	1.5
size (N)	62^3	62^3	62^3	100^3	100^3
Precision	10^{-5}	10^{-8}	10^{-9}	10^{-11}	10^{-11}
Relative gain	2.52	2.55	2.52	2.57	2.54

bandwidth (Mbit/s)	50	50	50	50	50
Power (GFlops)	1.5	1.5	1.5	1.5	1.5
size (N)	110^3	120^3	130^3	140^3	150^3
Precision	10^{-11}	10^{-11}	10^{-11}	10^{-11}	10^{-11}
Relative gain	2.53	2.51	2.58	2.55	2.54

relative gain. So, our objective running the algorithm in SimGrid is to obtain a relative gain greater than 1. A priori, obtaining a relative gain greater than 1 would be difficult in a local area network configuration where the synchronous GMRES method will take advantage on the rapid exchange of information on such high-speed links. Thus, the methodology adopted was to launch the application on a clustered network. In this configuration, degrading the inter-cluster network performance will penalize the synchronous mode allowing to get a relative gain greater than 1. This action simulates the case of distant clusters linked with long distance networks as in grid computing context.

Both codes were simulated on a two clusters based network with 50 hosts each, totalling 100 hosts. Various combinations of the above factors have provided the results shown in Table I. The problem size of the 3D Poisson problem ranges from $N = N_x = N_y = N_z = 62$ to 150 elements (that is from $62^3 = 238{,}328$ to $150^3 = 3{,}375{,}000$ entries). With the asynchronous multisplitting algorithm the simulated execution time is on average 2.5 times faster than with the synchronous GMRES one.

Note that the program was run with the following parameters:

SMPI parameters:

- HOSTFILE: Text file containing the list of the processors units name. Here 100 hosts;
- PLATFORM: XML file description of the platform architecture with the following characteristics:
  - 2 clusters of 50 hosts each;
  - Processor unit power: 1 GFlops or 1.5 GFlops;
  - Intra-cluster network bandwidth: 1.25 Gbit/s and latency: 50 μs;
  - Inter-cluster network bandwidth: 5 Mbit/s or 50 Mbit/s and latency: 20 ms;

Arguments of the program:

- Description of the cluster architecture matching the format <Number of clusters> <Number of hosts in cluster1> <Number of hosts in cluster2>;
- Maximum numbers of outer and inner iterations;
- Outer and inner precisions on the residual error;
- Matrix size N_x, N_y and N_z;
- Matrix diagonal value: 6 (see Equation (3));
- Matrix off-diagonal values: -1;
- Communication mode: asynchronous.

Interpretations and comments: After analyzing the outputs, generally, for the two clusters including one hundred hosts configuration (Tables I), some combinations of parameters affecting the results, have given a relative gain of more than 2.5, showing the effectiveness of the asynchronous multisplitting compared to GMRES with two distant clusters.

With these settings, Table I shows that after setting the bandwidth of the inter cluster network to 5 Mbit/s, the latency to 20 millisecond and the processor power to one GFlops, an efficiency of about 40 % is obtained in asynchronous mode for a matrix size of 62^3 elements. It is noticed that the result remains stable even if the residual error precision varies from 10^{-5} to 10^{-9}. By increasing the matrix size up to 100^3 elements, it was necessary to increase the CPU power by 50 % to 1.5 GFlops to get the algorithm convergence and the same order of asynchronous mode efficiency. Maintaining a relative gain of 2.5 and such processor power but increasing network throughput inter cluster up to 50 Mbit/s, is obtained with high external precision of 10^{-11} for a matrix size from 110^3 to 150^3 side elements.

VI. Conclusion

The simulation of the execution of parallel asynchronous iterative algorithms on large scale clusters has been presented. In this work, we show that SimGrid is an efficient simulation tool that has enabled us to reach the following two objectives:

1) To have a flexible configurable execution platform that allows us to simulate algorithms for which execution of all parts of the code is necessary. Using simulations before real executions is a nice solution to detect potential scalability problems.
2) To test the combination of the cluster and network specifications permitting to execute an asynchronous algorithm faster than a synchronous one.

Our results have shown that with two distant clusters, the asynchronous multisplitting method is faster by 40 % compared to the synchronous GMRES method which is not negligible for solving complex practical problems with ever increasing size.

Several studies have already addressed the performance execution time of this class of algorithm. The work presented in this paper has demonstrated an original solution to optimize the use of a simulation tool to run efficiently an iterative parallel algorithm in asynchronous mode in a grid architecture.

In future works, we plan to extend our experimentations to larger scale platforms by increasing the number of computing cores and the number of clusters. We will also have to increase the size of the input problem which will require the use of a more powerful simulation platform. At last, we expect to compare our simulation results to real execution results on real architectures in order to better experimentally validate our study. Finally, we also plan to study other problems with the multisplitting method and other asynchronous iterative methods.

Acknowledgment

This work is partially funded by the Labex ACTION program (contract ANR-11-LABX-01-01).

References

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

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

[3] J. Bahi, S. Contassot-Vivier, and R. Couturier, "Performance comparison of parallel programming environments for implementing AIAC algorithms," *Journal of Supercomputing*, vol. 35, no. 3, pp. 227–244, 2006.

[4] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw. Pract. Exper.*, vol. 41, no. 1, pp. 23–50, Jan. 2011.

[5] (2014) SimGrid website. [Online]. Available: http://simgrid.org/

[6] Y. Saad and M. H. Schultz, "GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems," *SIAM Journal on Scientific and Statistical Computing*, vol. 7, no. 3, pp. 856–869, 1986.

[7] H. Casanova, A. Legrand, and M. Quinson, "SimGrid: a generic framework for large-scale distributed experiments," in *Proceedings of the Tenth International Conference on Computer Modeling and Simulation*, ser. UKSIM '08. IEEE Computer Society, 2008, pp. 126–131.

[8] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, "Versatile, scalable, and accurate simulation of distributed applications and platforms," *Journal of Parallel and Distributed Computing*, 2014, to appear.

[9] (2014) Message Passing Interface MPI forum. [Online]. Available: http://www.mpi-forum.org/

[10] P. Bedaride, A. Degomme, S. Genaud, A. Legrand, G. S. Markomanolis, M. Quinson, M. Stillwell, F. Suter, and B. Videau, "Toward Better Simulation of MPI Applications on Ethernet/TCP Networks," in *PMBS13 - 4th International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*, Nov. 2013.

[11] P. Velho, L. Schnorr, H. Casanova, and A. Legrand, "On the Validity of Flow-level TCP Network Models for Grid and Cloud Simulations," *ACM Transactions on Modeling and Computer Simulation*, vol. 23, no. 4, Oct. 2013.

[12] A. Guermouche and H. Renard, "A First Step to the Evaluation of SimGrid in the Context of a Complex Application," in *19th International Heterogeneity in Computing Workshop (HCW)*. IEEE, Apr. 2010.

[13] P.-N. Clauss, M. Stillwell, S. Genaud, F. Suter, H. Casanova, and M. Quinson, "Single Node On-Line Simulation of MPI Applications with SMPI," in *Proc. of the 25th IEEE Intl. Parallel and Distributed Processing Symp (IPDPS)*. IEEE, May 2011, pp. 661–672.

[14] D. P. O'Leary and R. E. White, "Multi-splittings of matrices and parallel solution of linear systems," *SIAM Journal on Algebraic Discrete Methods*, vol. 6, no. 4, pp. 630–640, 1985.