# A Fine-grained Task Based Parallel Programming Paradigm of Gauss-Jordan Algorithm

Ling Shang, Maxime Hugues, and Serge G. Petiton
Computer Science Laboratory of Lille, University of Sciences and Technology of Lille, France
Grand Large Team, INRIA Futurs, France
Email: {ling.shang; maxime.hugues; serge.petiton}@lifl.fr

***Abstract*—Large scale matrix inversion has been widely used in many scientific research domains. As a classical method of large matrix inversion, block-based Gauss-Jordan (BbGJ) algorithm has aroused great concern among many researchers. Many people have developed parallel versions of BbGJ. But large granularity based task parallelism (intra-iterative data dependence based tasks parallelism) disables mapping more tasks on computing resources simultaneously. As a result, the performance of those parallel versions degrades when running on the Grid platform consisting of a lot of heterogeneous PCs and workstations. So this paper presents a fine-grained tasks based parallel BbGJ algorithm (Max-par BbGJ) in which both intra-iterative and inter-iterative based data dependences are considered. According to the analysis of data dependence in BbGJ, a global flag is adopted to control all the data dependence during the process of algorithm execution. The flag can be presented as a three-tuple. At the beginning, all the logical values of those three-tuples are set as false. When the logical value of the three-tuple becomes true, the tasks decided by the three-tuple can be executed immediately and simultaneously. Max-par BbGJ aims at achieving maximum parallelism of different parts of tasks. To evaluate its performance, YML a new high-level parallel programming tool, is introduced for its series of good features, such as components reuse, extreme ease-of-use and platform-independence. Grid environments are based on Grid'5000 platform in France. Experiments illustrate that the better performance can be achieved through making more tasks executed simultaneously in the Max-par BbGJ. The time needed using Max-par BbGJ can save about 30% than that using conventional one. At the same time, experiments also validate that though a little overhead existing, YML is still an acceptable and easy-to-use tool for scientific researchers especially for those non-professionals to make large scale computing.**

***Index Terms*—Gauss Jordan algorithm, programming paradigm, data dependence, formal description, Grid, YML**

## I. Introduction

The availability of powerful PCs and high speed network has changed the way of using computers in the last decade. A number of scientific applications traditionally performed on supercomputers, have been run on the Grid environment consisting of PCs and workstations. More and more scientific researchers show their interests on Grid for its lower costs, easy maintenances and high performance [1][13].

Large scale computing, especially large scale numerical algebra computing is always playing an important role in scientific computing. Large scale matrix inversion, as an important example of large numerical algebra computing can be used in weather prediction, long time climate evolution, aircraft design, DNA decoding, graphic transformation and so on. Sequential BbGJ algorithm is a classical method for matrix inversion. To get better performance, some parallel versions have been presented. Petiton shows its parallel version adaptive to MIMD[3]. His work focuses on finding an appropriate schedule method on MIMD vector processor to get higher performance. What our work differs from this work is that we will present a general parallel programming paradigm and an easy way to understand all the data dependence in the BbGJ algorithm. Melab et al give us its version tailoring to MARS which is a kind of network of workstations [4][5]. In his paper, he analyzes all the possible data dependence but his work only deal with the intra-iterative based data dependence without considering the inter-iterative based data dependence. Two kinds of parallelism (see Figure 3) which are inter-loop based parallelism and intra-loop based parallelism, in an iterative step are fully described in his paper. His parallel adaptive algorithm aims at dealing with dynamic changes of computing resources and scheduler can automatically fold/unfold tasks to computing resources according to their availabilities. Aouad et al present its intra-iterative based parallel version [7][14][15][16]. Aouad's work differ from work mentioned above is that he presents a general parallel programming paradigm. But his paradigm is only based on intra-iterative based data dependence. Ignoring the inter-iterative based data dependence causes the tasks between different iterative steps can't be executed concurrently, which restricts computing resources in Grid environment can be made full use of. To overcome the problem existed, analysis has been made on the sequential version of BbGJ algorithm. Some key characters of the BbGJ algorithm can be summarized as follows:

- The number of iterative step of the sequential BbGJ algorithm is equal to that of square root of *partitioned blocks of matrix* (sub-matrices).

Corresponding author : Serge G. Petiton.
Professor of computer science laboratory of Lille
Bat M3 59655 Villeneuve d'Asca cedex, France; Fax 03 28 77 85 37


doi:10.4304/jcp.5.10.1510-1519

- ✧ In the BbGJ algorithm, data dependence between sub-matrices plays an important role in deciding which parts of program can be executed concurrently.
- ✧ All the operations (data input and output) are on sub-matrix and the number of "data input" is same in each iterative step (see Figure 2).
- ✧ The goal of "data input" operation on sub-matrices in each iterative step is fixed, i.e. Operation '1', '3', '5' (marked in sequential BbGJ algorithm in section Ⅲ.A) work on the matrix A while Operation '4', '6', '7' work on the matrix B which is the inversion of matrix A ( see Figure 2).

Through the analysis, we can find that different operations (see operation '1' to '7' in sequential BbGJ) act on different matrices (the original matrix and its inversion matrix) and data input/output operation on sub-matrix play an important role in deciding which parts of algorithm can be executed in parallel. To the convenience of describing the relation of data input and output operation on sub-matrices, we regard those two matrices as an augmented matrix. Then a three-tuple (k,i,j) is used for controlling all the operations during the process of algorithm execution, in which, k represents the number of iterative step in BbGJ algorithm, i represents the row of the augmented matrix while j is the column of the matrix. Based on that, this paper presents a new method using a set of three tuples as global signals to control the data input/output operations on sub-matrix during the process of algorithm execution. Finally, a general parallel programming paradigm considering all kinds of data dependence is presented in this paper. In the following parts of paper, we call this paradigm as 'Max-par BbGJ' algorithm for its goals of chasing maximum degree parallelism of tasks.

To verify the performance of Max-par BbGJ, some experiments will be made. Here we will use YML [20][9][10][11] for its good features, such as easy-to-use, component reuse, Grid middleware independence. We will introduce YML in detail in next part. Here what we want to emphasize is that whatever kinds of middleware or programming environment is not key issue. You can use MPI or any other programming environments to make the same experiments, i.e. our paradigm is independent of any platform or programming environments. But through this paper, you will find that advantages are obvious for scientific researchers to make large scale computing using YML. We will demonstrate its advantage in detail in section Ⅳ. The platform used in our experiments is based on Grid'5000 in France.

The remainder of the paper is organized as follows: the second section introduces the platforms and middleware used in this paper. The data dependence in the BbGJ will be analyzed in section Ⅲ. Section Ⅳ shows us the new parallel programming paradigm: Max-par BbGJ algorithm. A version tailoring to YML is presented and advantages of using YML are described in section Ⅴ. Then several experiments will be made in Section Ⅵ. In seventh section, conclusions and future work are presented.

## II. Related Platform and Programming Environment

This section describes Grid'5000 platform which is test-bed of all those experiments in this paper. Then as a high-level parallel programming tool, YML is introduced in detail. OmniRPC as a backend of YML is also presented in this paper.

### *A. Grid'5000*

Grid'5000 [8] is national wide experimental set of clusters which provides a reconfigurable and highly controllable and monitorable infrastructure. It is composed of nine geographically distributed sites with 100 to 1000 heterogeneous nodes in each one. All the sites are interconnected by the French national research network RENATER. A set of control and monitoring tools allows users to make reservation (OAR), reconfiguration (Kadeploy [21]) and run experiments. Grid'5000 is not a production grid but an instrument that can be configured to work as a real test-bed on a nation wide scale. Many kinds of scientific researches at different levels in Grid, including fault tolerance, parallel programming, network protocols, middleware development/evaluation, scheduling and so on can be made in the Grid5000 platform.

### *B. YML*

YML [20][9][10][11] is a framework, developed at PRiSM laboratories in collaboration with Laboratoire d'Informatique Fondamentale de Lille (Grand Large Team, INRIA Futurs). The aim of YML is to provide users with an easy-of-use method to run parallel applications on different Grid platform. The framework can be divided into three parts which are end-users interface, frontend and backend. End-users interface is used to provide an easy-to-use and intuitive way to submit their applications and applications can be described using a workflow language YvetteML. Frontend is the main part of YML which includes compiler, scheduler, data repository, abstract component and implementation component. Abstract components and implementation components based on function can be reused. Backend is the part to connect different Grid and peer to peer middleware. See detail in Figure 1. The development of an YML application is based on components approach. Then we will discuss the three kinds of components in detail.

1. Abstract component: an abstract component defines the communication interface with the other components. This definition gives the name and the communication channels with other components. Each channel corresponds to a data input, data output or both and is typed. This component is used in the code generation step and to create the graph. This part of work can be developed by computer engineering in advance according to users' requirement.

2. Implementation component: an implementation component is the implementation of an abstract component. It provides the description of computations through YvetteML language. The implementation is done by using common languages like C or C++. They can have several implementations for a same abstract component. This part of work also can be developed by computer engineers in advance.
3. Graph component: a graph component carries a graph expressed in YvetteML instead of a description of computation. It provides the parallel and sequential parts of an application and the synchronization events between dependent components. It is very easy way for scientific researchers to develop their application. In this paper, we will take our version of BbGJ as example to explain it.

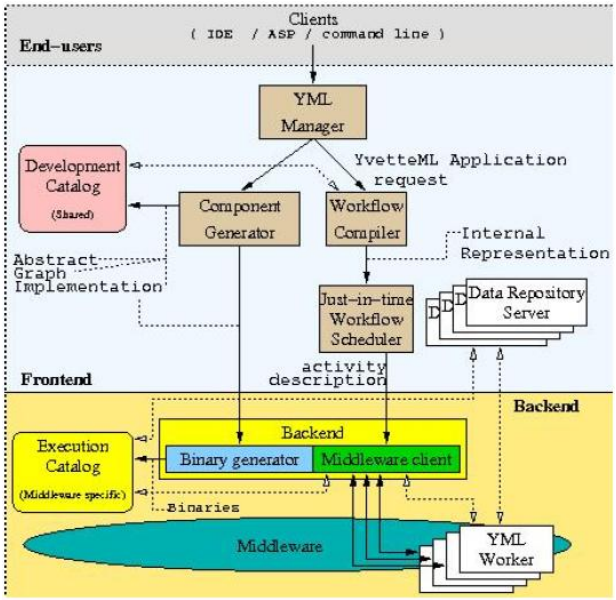


Figure 1. Framework of YML

Moreover, those three components are independent of middleware. So, to run an application on another grid environment with a different middleware, the user just needs compile each component for the middleware of his choice. OmniRPC as the backend of YML will be used in this paper.

### C. OmniRPC

OmniRPC[12] is a thread-safe grid RPC based on the Ninf facility for cluster and global computing environments. The remote libraries are implemented as executable programs in each remote computer, and OmniRPC automatically allocates remote library calls dynamically on appropriate remote computers to facilitate location transparency. It supports master-worker cluster and Grid applications.

## III. Data Dependence in the BbGJ Algorithm

### A. The Sequential Algorithm of Block-based Gauss-Jordan

The sequential BbGJ algorithm is a classical linear algebra method to get a large scale matrix inversion. The algorithm can be described as follows: Let A and B be two dense matrices of dimension N, and let B be the inverse of A, i.e. AB=BA=I. Let A and B be partitioned into a matrix of q×q blocks of dimension n which n = N/q.

**Sequential BbGJ Algorithm**

*Input* : $A,B \leftarrow I(n,n),q$
*Output* : $B = A^{-1}$

for $k$ = 1 to $q$ do
$A^k_{k,k} \leftarrow (A^{k-1}_{k,k})^{-1}$ (1)
$B^k_{k,k} \leftarrow A^k_{k,k}$ (2)
  for $j$=$k$+1 to $q$ do
    $A^k_{k,j} \leftarrow A^k_{k,k} A^{k-1}_{k,j}$ (3)
  end for
  for j=1 to k-1 do
    $B^k_{k,j} \leftarrow A^k_{k,k} B^{k-1}_{k,j}$ (4)
  end for
  for $j$=$k$+1 to $q$ do
    for $i$=1 to $q$ and $i \neq k$ do
      $A^k_{i,j} \leftarrow A^{k-1}_{i,j} - A^{k-1}_{i,k} A^k_{k,j}$ (5)
    end for
  end for
  for $j$=1 to $k$-1 do
    for $i$=1 to $q$ and $i \neq k$ do
      $B^k_{i,j} \leftarrow B^{k-1}_{i,j} - A^{k-1}_{i,k} B^k_{k,j}$ (6)
    end for
  end for
  for $i$=1 to $q$ and $i \neq k$ do
    $B^k_{i,k} \leftarrow -A^{k-1}_{i,k} A^k_{k,k}$ (7)
  end for
end for

### B. Parallelism in the Algorithm

Through the sequential BbGJ algorithm, we can find that each iterative step of algorithm is made up of five loops (the third, fourth, fifth, sixth and seventh operation marked in sequential BbGJ algorithm) and two other operations (the first and second ones in sequential BbGJ algorithm). In the iterative step k (k = 1,…,q), the first operation is used to get the inverse of pivot block $A^{k-1}_{k,k}$ ; the second operation will assign the sub-block of matrix A to corresponding sub-block of matrix B; the third operation computes the blocks of the row k of matrix A with subscripted variables j above k while the fourth operation acts on the row k of matrix B with subscripted variables j below k; the fifth operation calculates the blocks of all columns of the matrix A with subscripted variables i above and below k and subscripted variables j above k; while the sixth operation calculates the blocks of all columns of the matrix B with subscripted variables i above and below k and subscripted variables j below k. At last the seventh operation is used to compute the blocks of the column number k of matrix B except $B_{k,k}$. Refer to Figure 2.

From the sequential BbGJ algorithm, we can also know that there are q iterative steps in the algorithm. In

each iterative step, there are total 7 operations (marked in the sequential BbGJ algorithm). Operations '3' to '7' are based on loop. And the intra loop based parallelism belongs to SPMD paradigm (single program multi data). While inter-loops based parallelism is determined by data dependence between loops. For example operation '3' must wait the result of operation '1', i.e. inter-loops based parallelism is to deal with concurrent or sequence execution between different operations in an iterative step. Another case of data dependence happens between different iterative steps of algorithm.

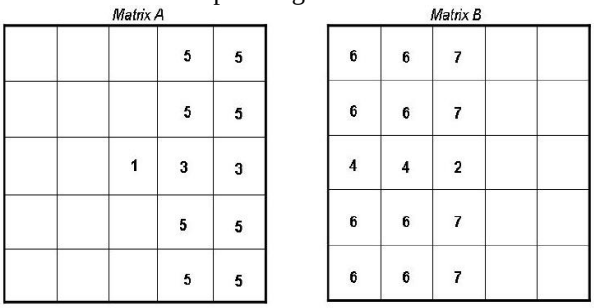


Figure 2. Operations in the 3rd Iterative Step with q is 5

So, the parallelization of the sequential BbGJ algorithm consists of exploiting two kinds of parallelism: the inter-iteratives based parallelism and the intra-iterative based parallelism. The intra-iterative parallelism aims at exploiting the parallelism involved in each of the five loops (operation '3' to operation '7' in sequential BbGJ algorithm). It falls into two categories: the inter-loops parallelism and the intra-loop parallelism. See Figure 3.

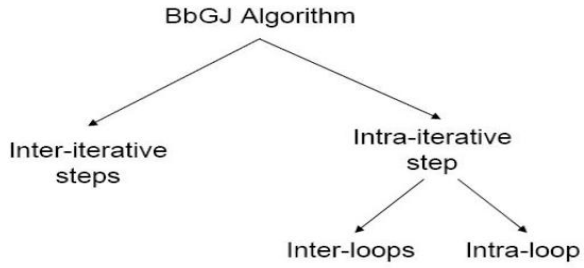


Figure 3. Parallelism in the Block-based Gauss-Jordan Algorithm

Next, we will analyze those parallel relationships one by one.

### C. Intra-iterative Based Parallelism

#### *C.1 The Inter-loops Based Parallelism:*

The inter-loops parallelism refers to operations between loops. There are dependable relationships which determine when those operations between loops can be executed. In the sequential BbGJ algorithm, we identify three ones: 1) the second, third, fourth and seventh operation (see the number marked in sequential BbGJ algorithm) can't be executed before knowing the value of block $A^k_{k,k}$; 2) the loop of the fifth operation depends on the third one because its input (the block $A^k_{k,j}$ ) is the output of the third operation; 3) the sixth operation is dependent on the fourth operation because its input (the block $B^k_{k,j}$) is the output of the fourth operation.

#### *C.2 The Intra-loop Based Parallelism:*

The intra-loop parallelism consists of executing each of the five loops (operations '3', '4', '5', '6' and '7') of the algorithm in parallel. They all belong to SPMD (single program multi data) paradigm. The work units for these loops are of two kinds: matrix product and matrix triadic respectively $X \times Y$ and $Z\text{-}X \times Y$, where X, Y and Z are sub-matrix blocks. Moreover, to get higher performance in a certain platform, the granularity of tasks is very important. Here we can balance the computing time and communication time through choosing appropriate block-size of sub-matrix. The experiment part of this paper will show us the influence on the efficiency of algorithm caused by changing the block-size and block-count of matrix.

### D. Inter-iteratives Based Parallelism

Next, we will take the data-dependence between step k and step k+1 as example to show the inter-iterative parallelism in the BbGJ algorithm. Getting the value $A_{k,j}$ (k=1…q; j=k+1…q) ( respectively $B_{k,j}$ in which k=1…q; j=1…k-1) of k+1 step in the third operation (respectively the fourth) needs get the value of block $A_{k,j}$ (k =1…q; j=k+1…q) ( respectively $B_{k,j}$ in which k=1…q; j=1…k-1) of k step. As to the fifth operation, we must know that block $A_{i,j}$ (i=1…q and i≠k; j=k+1…q) and $A_{i,k}$ (i=1…q and i≠k; k=1…q) of k step before we compute the value $A_{i,j}$ (i =1…q and i≠k; j= k+1…q) of k+1 step. The situation in the sixth operation is just like that in the fifth, we can compute the value $B_{i,j}$ (i=1,…,q and i≠k; j=1…k-1) of step k+1 after we getting the value $B_{i,j}$ (i =1…q and i≠k; j=1…k -1) and $A_{i,k}$ (i =1…q and i≠k; k=1…q) of step k. As to the operation '7', to get the value $B_{i,k}$ (i=1…q and i≠q; k=1…q) of k+1 step, we must know the value $A_{i,k}$ (i=1…q and i≠q; k=1…q) of step k.

### E. Description of Data Dependence

We have analyzed all the possible data dependence in the BbGJ algorithm, but it is hard to understand and find the regularity. So in this part an intuitive method (using tables) is presented to express data dependence.

#### *E.1 Description of Intra-iterative Based Data Dependence:*

Then we can describe the data dependence in sequential BbGJ algorithm using Figure 4 and Figure 5. In those figures the italic font (in blue) represents the read operation (a value will be read from the block as data output) and black font with underline donates the write operation (a value will be written into the block as data input). For example, the italic font '5' in the first row, second column of matrix A in Figure 4 represents that: to this block, the first operation is '5' (see the number marked in sequential BbGJ algorithm) which is a read operation, i.e. operation '5' will read the data from this block as its input. The same way we can know that the black font with underline '1' in the second row, second column of matrix A in Figure 4 represents that: the operation '1' will assign a new value to this block. Next we will take the block in the second row, first column of

matrix B in Figure 4 as example to show all the related operations executed in this block. From the Figure 4, we can know three operations executed in this block, of which italic fonts ‘4’ and ‘6’ are read operations and black font with underline ‘4’ is write operation. To this block, operations will be executed from up to down, i.e.

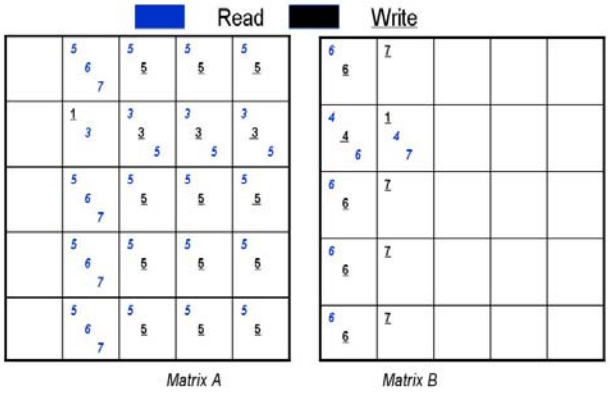


Figure 4. Blocks Based Operation at the 2ed Iterative Step with q is 5.

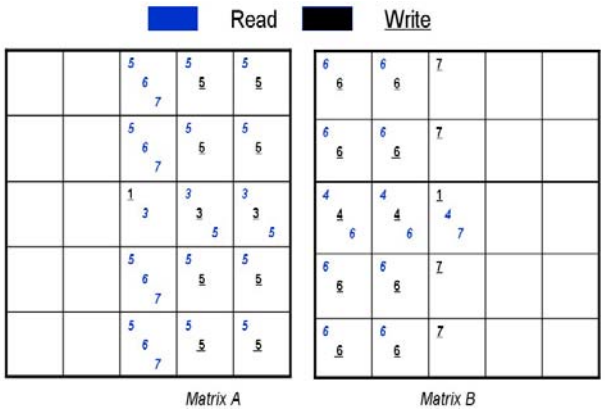


Figure 5. Blocks Based Operation at the 3rd Iterative Step with q is 5.

the read operation ‘4’ will read from this block as its data input at first, then write operation ‘4’ assigns a new value to this block as its data output. Finally, read operation ‘6’ will read the value of this block as its data input. After that, no more operations are made on this block in this iterative step. Figure 4 shows us all the operations happened at the second iterative step with q is 5 in the BbGJ algorithm. We can also get all the operations information at the third iterative step with q is 5 from the Figure 5. Through the analysis, we can summarize the intra-iterative based data dependence using Figure 6.

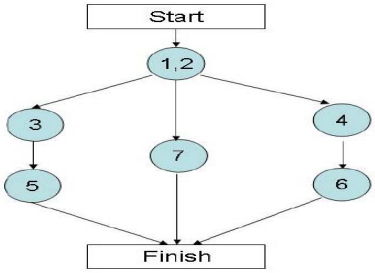


Figure 6. Intra-iterative Based Data Dependence.

***E.2 Description of Inter-iteratives Based Data Dependence:***

We have presented the intra-step data dependence through the Figure 4 and Figure 5. Now we will analyze the inter-iteratives based data dependence. To do that, we define “directed arc” as follows: a directed arc from block “X” to block “Y” (X, Y are blocks of matrix) signifies that the operation on block “Y” can’t be executed before the operation on block “X” finishes. See directed arc in Figure 7. There are two operations in ‘loop 1’, which are ‘2.5’ and ‘3.1’. ‘2.5’ represents the fifth operation in second iterative step of BbGJ, while ‘3.1’ stands for the first operation in the third iterative step of BbGJ. ‘Loop 1’ in the Figure 7 shows us data dependence between operation ‘1’ at second iterative step and operation ‘5’ at the third iterative step. And it means the operation ‘5’ at third iterative step can’t be executed until the finishes of operation ‘1’ at second iterative step. In the same way, we can analyze the data dependence represented by ‘loop n’ (n is 1, 3, 4, 5, 6, 7). Figure 7 shows us all the inter-iteratives based data dependence existing between step 2 and step 3 with q is 5.

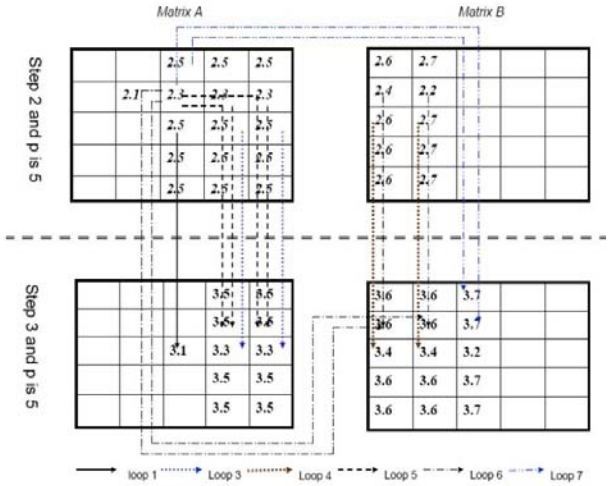


Figure 7. Inter-iteratives Based Data Dependence.

***E.3 Description of all the Data Dependences:***

From the Figure 6, we can know all the intra-iterative based data dependence and Figure 7 show us all the inter-

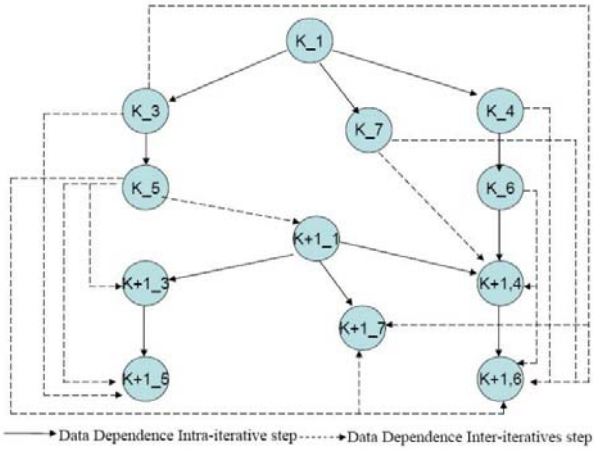


Figure 8. Data Dependence in the Algorithm.

iteratives based data dependence. Then we can summarize all the data dependence existing in BbGJ algorithm using the Figure 8.

## IV. Parallel Programming Paradigm on BbGJ Algorithm

From Figure 4 and Figure 5, we can find that each iterative step of the algorithm has the same number of write operations and the number is q. To the matrix A ,the write operation on it are operations '1', '3' and '5', while operations '4', '6' and '7' act on matrix B. It is the similarities existing in write and read operation on matrix A and matrix B that we can take matrix A and matrix B together into account as an augmented matrix.

***Definition 1***: Let

AB(i,j) = A(i,j) i=1…q; j=1…q

AB(i,j+q) = B(i,j) i=1…q; j=1…q

$AB^k_{i,j}$ represents the block $AB_{i,j}$ at step k for k = 1,…, q. In the BbGJ algorithm, one block ($AB^k_{i,j}$) is modified once and only once per iterative step. Let's use a three-tuple (k,i,j) to represent $AB^k_{i,j}$ . Then we can use the three-tuple (k,i,j) {k,i=1,...,q and j=k+1,...,k+q} to mark the status of sub-matrix's operations (operation '1' to '7' in sequential BbGJ algorithm), i.e. the three-tuple (k,i,j) means that at the iterative step k, the operation will be made on the row i and column j of the augmented matrix $AB_{i,j}$.

For a fixed step, a set of three-tuples can be utilized to represent the status of operations on augmented matrix AB. Consequently we can use those three-tuples as global signals to control all the operations decided by data dependence in the algorithm.

***Definition 2***: Let the binary relation be defined as follows: X ≻ Y: (X, Y is the block of matrix) if and only if there exists an edge from X to Y in the Figure 8. The data dependence can thus be represented by the three-tuple presented in definition 1. For all the data dependence existing in Figure 8, we have:

Intra-iterative based data dependence can be written as follows:

- (k, k, k)≻(k, k, j) k=1… q; j=k+1… q
- (k, k, k)≻(k, k, j+q) k=1… q; j=1…k-1
- (k, k, j)≻(k, i, j) k=1…q ; j=k+1…q ; i=1…q and i≠k
- (k, k, j+q)≻(k, i, j+q) k=1… q; j=1…k-1 ; i=1…q and i≠k
- (k, k, k)≻(k, i, k+q) k=1…q; i=1…q and i≠k

Inter-iteratives based data dependence can be summarized as follows:

- (k-1, k, k)≻(k, k, k) k=2… q
- (k-1, k, j)≻(k, k, j) k=2…q; j= k+1…q
- (k-1, k, j+q)≻(k, k, j+q) k=2… q; j=1…k-1
- (k-1, i, k)≻(k, i, j) k=2…q; j=k+1…q; i=1…q and i≠k
- (k-1, i, j)≻(k,i, j) k=2…q; j= k+1… q; i=1…q and i≠k
- (k-1, i, j+q)≻(k, i, j+q) k=2…q; j=1… k-1; i=1… q and i≠k
- (k-1, i, k)≻(k, i, j+q) k=2… q; j=1…k-1; i=1… q and i≠k
- (k-1, i, k)≻(k, i, k+q) k=2…q; i=1…q and i≠k

Now we will summarize binary relations obtained above and present our new parallel programming paradigm of BbGJ algorithm.

**Max-Par BbGJ Algorithm**

**Input** : $A,B \leftarrow I(n,n), q$

the logical value of (0,1,1) is set to be true;
for(i=1;i<=q;i++) the logical value of (0,1,i) is set to be true;
for(i=1;i<=q;i++) the logical value of (0,i,1) is set to be true;
for(i=1;i<=q;i++)
  for(j=1;j<=q;j++)
    the logical value of (0,i,j) is set to be true;

**Output** : $B = A^{-1}$

---

**if** the logical value of (k, k, k) is true
**then** $A^k_{k,k} \leftarrow (A^k_{k-1,k})^{-1}$; (k=1,...,q; j=k+1,...,q )
  the logical value of (k, k, j) is set to be true;
**end if**

//

**if** the logical value of both (k, k, k) and (k-1,k,j) are true
**then** $A^k_{k,j} \leftarrow A^k_{k,k}A^{k-1}_{k,j}$ (k=2,...,q; j=k+1,...,q)
  the logical value of (k, k, j) is set to be true;
**end if**

//

**if** the logical value of both (k, k, k) and (k-1,k,j+q) are true
**then** $B^k_{k,j} \leftarrow A^k_{k,k}B^{k-1}_{k,j}$ (k=2,...,q; j=1,...,k-1)
  the logical value of (k, k, j+q) is set to be true;
**end if**

//

**if** the logical value of (k, k, j) and (k-1,i,k) and (k-1,i,j) all are true
**then** $A^k_{i,j} \leftarrow A^{k-1}_{i,j} - A^{k-1}_{i,k}A^k_{k,j}$ (k=2,...,q; j=k+1,...,q; i=1,...,q and $i \neq k$)
  the logical value of (k, i, j) is set to be true;
**end if**

//

**if** the logical values of (k, k, j+q) and (k-1,i,j+q) and (k-1,i,k) all are true
**then** $B^k_{i,j} \leftarrow B^{k-1}_{i,j} - A^{k-1}_{i,k}B^k_{k,j}$ (k=2,...,q; j=1,...,k-1; i=1,...,q and $i \neq k$)
  the logical value of (k, i, j+q) is set to be true.
**end if**

//

**if** the logical value of both (k, k, k) and (k-1,i,k) are true
**then** $B^k_{i,k} \leftarrow -A^{k-1}_{i,k}A^k_{k,k}$ (k=2,...,p; i=1,...,p and $i \neq k$)
  the logical value of (k, i, k+q) is set to be true.
**end if**

---

***Note to the Max-par BbGJ algorithm: the signal "//" is meaningless in the algorithm. We just want to use this signal to divide the whole program into several sub-program sections. And those subprogram sections can be executed in parallel when their conditions are met.***

### *A. Characteristic of Max-Par BbGJ Algorithm*

Firstly, Max-par BbGJ is a parallel programming paradigm for BbGJ. It can tailor to any programming environments very easily.

Secondly, the key feature of Max-par BbGJ Algorithm is that it can ensure all possible parallel parts in the algorithm can be executed in parallel.

The old parallel versions of BbGJ [2][6][4][5] [7][14][15][16] [18] all are intra-iterative based parallel algorithm, while our work, all the data dependence of operations are considered. Intra-iterative based parallelism means the operations in next iterative can't be executed before any operation in this iterative step

finishes, i.e. if the operation '5' finishes, but operation '1' in next iterative step must wait for operation '7' and operation '6' until they both finish (see Figure 6). But to Max-par BbGJ algorithm, operation '1' in next iterative can be executed immediately once the operation '5' finishes (see Figure 8). So less wait/synchronization time is needed in Max-par BbGJ algorithm. As a result, the greatest degree of parallelism on tasks is achieved in Max-par BbGJ algorithm.

## V. MAX-PAR BBGJ ALGORITHM ADAPTIVE TO YML

The reason we use grid middleware YML is for its two good features: algorithm independence and platform independence. These two features make YML become an easy-to-use middleware for non professional programmer and user can utilize different platforms without changing theirs codes. Detail explanation will be made in the following sectors.

### A. Algorithm Independence

To make the BbGJ algorithm tailor to YML, we need deal with four basic operations in the algorithm which are:

1. *inversion*: to invert one matrix (operation '1').
2. *prodMat*: to compute the matrix multiplication (operation '3' and '4').
3. *mProdMat*: to compute the negative of matrix multiplication (operation '7').
4. *ProdDiff*: to compute the difference between a matrix and matrix multiplication (operation '5' and '6')

For experiments based on YML environment, we can develop all the related abstract and implementation components for these basic operations in advance and this part of work has nothing to do with data dependence in the algorithm. Just like presented above, four basic functions can be developed in advance. These functions just are basic operations and have nothing to do with algorithm itself. We can apply these basic functions to other algorithm without any change. For example when you want to compute two matrices multiplication, you can reuse the developed components (for example 'prodMat' in this algorithm). Those developed components can be reused without any change. We call this feature of YML as algorithm independence.

### B. Platform Independence

You can choose middleware supported by YML to harness different kinds of computing resources. For example, OmniRPC can be used for harnessing computing resources in cluster, cluster of clusters or grid. XtremWeb[17] can deal with volunteer computing resources (desktop grid). So users can harness computing resources to make large scale computing in different platform without changing theirs codes. Their work is just to write a graph component according to the data dependence using YvettML language, which is a description language based XML and it is very easy to learn even to non-professional programmer. The graph components are compiled on the client node and tasks are scheduled to computing resources automatically. During this process, the data repository server sends the data needed to computing resources and collects the final results from them automatically.

### C. Adaptive BbGJ Algorithm

The Max-par BbGJ algorithm adaptive to YML can be written as follows:

**Adaptive BbGJ Algorithm**
**Input**: $A,B \leftarrow I(n,n)$, *blockcount*, *blocksize*

```
  notify(flag[-1][0][0]);
 par
  par(i:=1; blockcount-1)
    do
      notify(flag[-1][0][i]);
    enddo
 //
  par(i:=1; blockcount-1)
    do
      notify(flag[-1][i][0]);
    enddo
 //
  par(i:=1; blockcount-1) (j:=1; blockcount-1)
    do
      notify(flag[-1][i][j]);
    enddo
 end par
```

**Output**: $B=A^{-1}$

```
 par
    wait(flag[k-1][k][k]);
    compute inversion(A[k][k],B[k][k],blocksize,blocksize);
    compute EvalMat(B[k][k],A[k][k],blocksize);
    notify(flag[k][k][k]);
    notify(flag[k][k][k+blockcount]);
 //
    wait(flag[k][k][k] and flag[k-1][k][j]);
    compute prodMat(A[k][k],A[k][j],blocksize);
    notify(flag[k][k][j]);
 //
    wait(flag[k][k][k] and flag[k-1][k][j+blockcount]);
    compute prodMat(A[k][k],B[k][j],blocksize);
    notify(flag[k][k][j+blockcount]);
 //
    wait(flag[k][k][k] and flag[k-1][i][k]);
    compute mProdMat(A[i][k],A[k][k],B[i][k],blocksize);
    notify(flag[k][i][k+blockcount]);
 //
    wait(flag[k][k][j] and flag[k-1][i][k] and flag[k-1][i][j]);
    compute prodDiff(A[i][k],A[k][j],A[i][j],blocksize);
    notify(flag[k][i][j]);
 //
    wait(flag[k][k][j+blockcount] and flag[k-1][i][k] and flag[k-
1][i][j+blockcount]);
    compute prodDiff(A[i][k],B[k][j],B[i][j],blocksize);
    notify(flag[k][i][j+blockcount]);
 endpar
```

Through comparing the Max-par BbGJ with adaptive BbGJ, we can find YML provides a very easy-to-use way to develop parallel algorithm. As to those components used in the algorithm, those parts can be done by computer engineer. And once they have been developed, they can be reused very easily. Of course a little overhead exists when using YML. In the next part, experiments will show us the overhead is acceptable comparing to cost of developing a new parallel algorithm adaptive to new programming environment.

## VI. EXPERIMENTS

Experiments can be divided into two parts. One is for verifying the performance of our new parallel programming paradigm. Here we choose the parallel versions provided by Aouad [7][14][15][16] as comparative object. Another is to evaluate the overhead of YML. In order to state conveniently we call our new algorithm as "Max-par BbGJ" (maximum-degree parallelism in the BbGJ algorithm) and Aouad's version of BbGJ as "Par-par BbGJ" (partial degree parallelism). All those experiments will be made based on Grid'5000[8] platform. Computational resources can be described as follows (see table1):

**Table 1.** Resources in Grid'5000

| Site | cluster | Nodes | CPU/Memory |
|---|---|---|---|
| Nancy | grelon | 120 | 2 × Inter xeon, 1.6GHz/2GB |
| Nancy | grillon | 47 | 2 × AMD opteron, 2GHz/2GB |
| Rennes | paravent | 99 | 2 × AMD opteron, 2GHz/2GB |
| Rennes | paraquad | 64 | 2 × AMD opteron, 2.2GHz/2GB |
| Lyon | Sagittaire | 70 | 2 × AMD opteron, 2.4GHz/2GB |
| Bordeaux | Bordereau | 93 | 2 × AMD opteron 2.6 GHz/2GB |

The first three experiments will be made by comparing the Max-par BbGJ with the Par-par BbGJ, through which good performance of Max-par BbGJ can be presented. Another experiment will be made by comparing the time consumed in YML+OmniRPC environment with that in OmniRPC environment, through which conclusion can be made that: YML is still a good solution for users to make large scale numerical computing.

## A. *The Performance of Max-par BbGJ*

### *A.1 Block-size Changed and Block-count Fixed:*

Experiment motivation: test the performance of Max-par BbGJ with block-size and fixed block-count.

Experiment environments: 20 nodes × 2= 40 cores used of cluster bordereau in Bordeaux site, France.

Experiment data: the block-count of matrix is 4×4 (fixed) then we change the block-size of sub-matrix from 50 × 50 to 1500 ×1500. Assume $B_k = \sqrt{block\_size}$ .

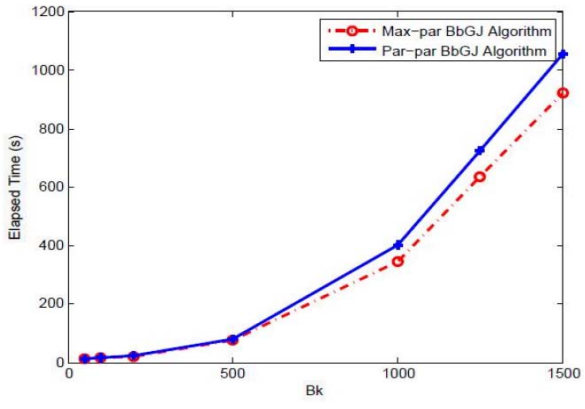


Figure 9. Elapsed Time with Block-size Changing.

From the Figure 9, we can find when the block-size is not large, the elapsed time of Max-par BbGJ is almost the same with that of Par-par BbGJ. The reason is that when the block-size is small, communication time plays an important role in the overall consumed time. Less time for synchronization is needed and the advantage of Max-par BbGJ algorithm is not obvious. But when the block-size is more than 100×100, the performance of Max-par BbGJ is better than Par-par BbGJ and with the increase of block-size, the advantage of Max-par BbGJ becomes more obvious. The reason is that, less wait time is needed in Max-par BbGJ algorithm comparing to that in Par-par BbGJ algorithm. (refer to analysis on section 4.1)

### *A.2 Block-size Changed and Block-count Changed:*

Experiment motivation: test the performance of Max-par BbGJ algorithm with different block-size and different block-count.

Experiment environments: 100 nodes× 2= 200 cores used of cluster Grelon in Nancy site.

Experiment data: the block-size used are 500×500, 1000×1000, 1500×1500. block-count used are 2×2, 3 ×3, 4×4, 5×5, 6×6, 7×7, 8×8.

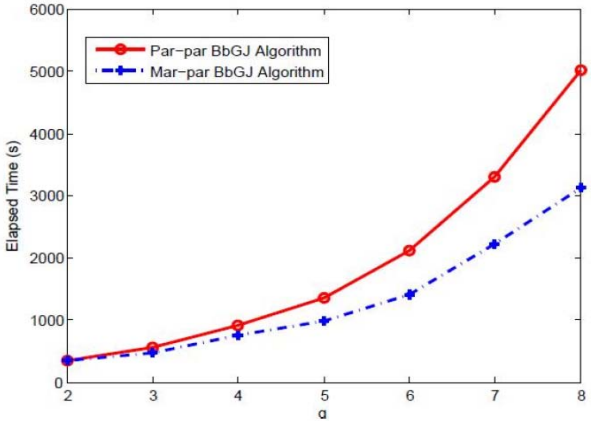


Figure 10. Elapsed Time with Block-count Changing.

The Figure 10 shows us the relationship between the elapsed time and the different block-counts of matrix. The block-size of sub-matrix is fixed on 1500×1500. We can know with the increase of block-count, the better performance can be obtained in Max-par BbGJ algorithm. The reason is that with the increase block-count of matrix, the number of iterative steps augments. More wait time is needed to synchronize at the end of each iterative step in the Par-par BbGJ, while no wait time is needed in Max-par BbGJ Algorithm. The same conclusion can be obtained using different block-size of sub-matrix. Next we will focus on the time difference between Max-par BbGJ and Par-par BbGJ with changing the block-size and block-count of sub-matrix.

The Figure 11 (Assume $B_k = \sqrt{block\_size}$ ) shows us the general trends of time difference with changing block-size and block-count. With the same block-size, the time difference becomes larger with increasing block-count of sub-matrix. The reason is analyzed through Figure 10. Another trend is that the time difference also becomes larger with the increase of block-size of sub-matrix. The reason is that the larger dimension of sub-matrix will cost more computing time. As a result, more wait time is needed to synchronize in Par-par BbGJ. Through the Figure 11 we can know the gap becomes larger with the increase of block-count and block-size. What we want to emphasize here is when the dimension of matrix is very large (whether through adding block-

count or enlarging block-size of sub-matrix), the advantage of Max-par BbGJ is more obvious

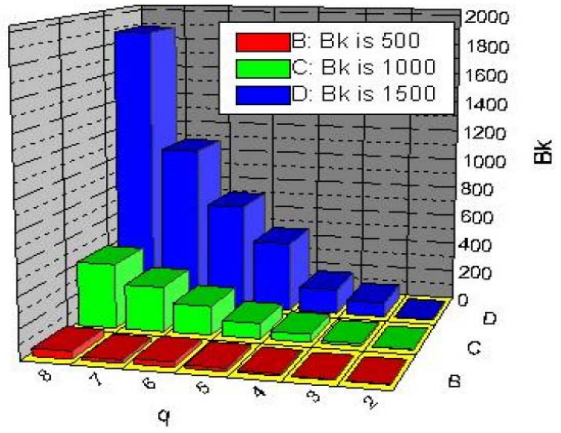


Figure 11. Elapsed Time with Block-count and Block-size Changing.

### *A.3 Performance in the Grid Environment:*

Experiment motivation: test the performance of Max-par BbGJ algorithm in the grid environment.

Experiment environments: 100 nodes× 2= 200 cores used. Grelon cluster, Nancy site; Paravent cluster and Paraquad cluster in Rennes site; Sagittaire cluster in Lyon site; Bordereau cluster in Bordeaux site. 20 nodes × 2 cores=40 cores are used in each cluster.

Experiment data: the block-size of matrix is 1500× 1500 (fixed) then we change block-count q from 2 to 8.

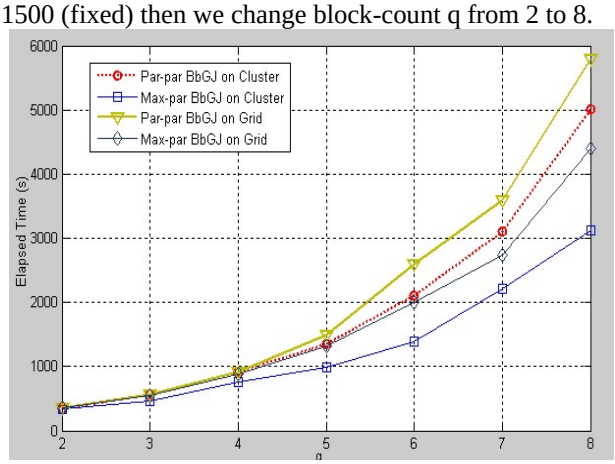


Figure 12. Elapsed Time in Different Environments.

From the Figure 12, we can know clearly that the same conclusion can be made in the Grid environment, and the reason is the same. But the elapsed time is longer on Grid than that on cluster, which is determined by the character of Grid environments (lower speed network than cluster, more complex schedule stratagem than cluster). An interesting case is that computation time of Par-par BbGJ based on cluster is longer than that of Max-par BbGJ on Grid. Two reasons can explain that. On one hand, the performance of Max-par BbGJ is far better than that of Par-par BbGJ. On the other hand, it is that high speed network between sites in Grid'5000 platform makes Max-par BbGJ algorithm have better performance.

### *B. Advantages of YML*

Experiment motivation: Here we want to present YML framework is a good solution for non-professional scientific researchers though a little overhead existing.

Experiment environments: Max-par BbGJ and Par-par BbGJ algorithm on YML framework+OmniRPC and Par-par BbGJ algorithm on OmniRPC. The reason is that complexity of data dependence in Max-par BbGJ algorithm makes it hard to program using OmniRPC directly.

Experiment data: we set the block-size of matrix is 1000×1000. We will change the block-count q from 2 to 8.

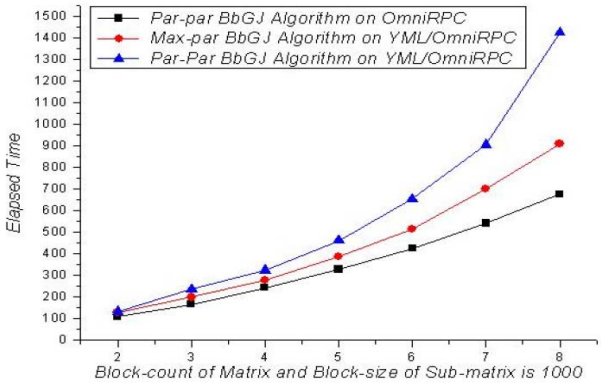


Figure 13. Performance of the Framework.

From the Figure 13, we can know the elapsed time using Max-par BbGJ algorithm with YML Framework is a little more than that of Par-par BbGJ with OmniRPC. But it is very difficult for people to tailor the Max-par BbGJ algorithm to the OmniRPC environment, even to professional programmer. To YML framework, user only need write a graph component to describe the data dependence for Max-par algorithm. Other works, such as developping data type, abstract component and implementation component can be done in advance by professional computer programmer. Once they are developed, all those codes can be reused (see detail in section Ⅴ). For example, in the paper [7][14][15][16], the Par-par BbGJ algorithm adaptive to YML and its matrix data type, matrix inversion, and matrix product components have been developed. When you want to tailor Max-par BbGJ algorithm, the only thing you need to do is to develop graph component. It is very easy because the adaptive BbGJ algorithm is very similar to Max-par BbGJ algorithm. But it is difficult for user to develop/reuse the codes for a special middleware environment such as OmniRPC. You have to rewrite the complex program to adapt to the new algorithm, even some common operations existing between those algorithms. So facing huge costs (time and money cost to develop), YML framework is acceptable for end user, especially to non-computer professional users and the overhead is small within the acceptable level.

## VII. CONCLUSION

This paper presents a fine-grained tasks based parallel BbGJ algorithm (Max-par BbGJ) in which both intra-iterative and inter-iterative based data dependences are considered. Through analyzing data dependence in BbGJ, a global flag (three tuple) is adopted to control all the data dependence during the process of algorithm execution. Max-par BbGJ achieved maximum parallelism of different parts of tasks. Experiments also testify that Max-par BbGJ can gain higher performance than Par-par BbGJ whether in cluster environment or in Grid environment. The improvement becomes larger with increasing block-count or block-size of sub-matrix (Figure 9, 10). When the dimension of matrix is very large (dimension of matrix in our experiment is $1500 \times 1500 \times 8 \times 8$, Figure 11), the improvement can be achieved more than 30%. Experiment also verifies that though a little overhead existing, YML is still a good choice to make a large scale computing for its many good features, such as algorithm independence and platform independence.

## ACKNOWLEDGMENT

Thanks to French Embassy in China and CROUS Lille which support Ling Shang to make further research in France. Here we also thanks Grid'5000 platform, YML, OmniRPC for their availabilities.

## REFERENCES

[1] BRUCE M. BOGHOSIAN, PETER V. COVENEY: scientific application of Grid computing. *COMPUTING IN SCIENCE ENGINEERING 2005 IEEE Co-published by the IEEE CS and the AIP*. Journal.

[2] Maxime Hugues and Serge G. Petiton: A Matrix Inversion Method with YML/OmniRPC on a Large Scale platform. VECPAR'08, Toulouse, France, 24-27 June 2008.

[3] S. Petiton, Parallelization on an MIMD computer with realtime Scheduler, *Aspects of Computation on Asynchronous Parallel Processors*, North Holland, 1989.

[4] N. Melab, E. Talbi, and S. Petiton, A parallel adaptive version of the block-based gauss-jordan algorithm, *in IPPS/SPDP, 1999*, pp. 350-354.

[5] N. Melab, E.-G. Talbi, and S. G. Petiton. A parallel adaptive gauss jordan algorithm. *The Journal of Supercomputing*, 17(2):167-185, 2000.

[6] M. Hugues. Algorithmique scientifique distribu grande chelle et programmation yml sur cluster de clusters. *Masters thesis*, 2007.

[7] L. M. Aouad and S. G. Petiton. Parallel basic matrix algebra on the grid5000 large scale distributed platform. In *CLUSTER, 2006.*

[8] F. Cappello, E. Caron, M. J. Dayde, F. Desprez, Y. Jegou, P. V.-B. Primet, E. Jeannot, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R.Namyst, B. Quetier, and O. Richard. Grid5000: a large scale and highly reconfigurable grid experimental testbed. In The *6th IEEE/ACM International Conference on Grid Computing*, pages 99-106, 2005.

[9] O. Delannoy, N. Emad, and S. G. Petiton.Workflow Global Computing with YML. In The *7th IEEE/ACM International Conference on Grid Computing*, pages 25-32, 2006.

[10] O. Delannoy and S. Petiton. A Peer to Peer Computing Framework: Design and Performance Evaluation of YML. In *Third International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks*, pages 362-369. IEEE Computer Society Press, 2004.

[11] O. Delannoy, YML: A scientific Workflow for High Performance Computing, Ph.D. Thesis, Septembre 2008, Versailles

[12] M. Sato, T. Boku, and D. Takahashi. OmniRPC: a Grid RPC system for Parallel Programming in Cluster and Grid Environment. In The *3rd IEEE International Symposium on Cluster Computing and the Grid*, pages 206-214, 2003.

[13] L. Shang, Z.Wang, X. Zhou, X. Huang, and Y. Cheng. TM-DG: a trust model based on computer users daily behavior for desktop grid platform. In *CompFrame 07: Proceedings of the 2007 symposium on Component and framework technology in high-performance and scientific computing*, pages 59-66, NY, USA, 2007. ACM.

[14] L. M. Aouad, S. Petiton, and M. Sato, Grid and Cluster Matrix Computation with Persistent Storage and Out-of-Core Programming, in The *2005 IEEE International Conference on Cluster Computing*, September 2005. Boston, Massachusetts, 2005.

[15] Serge G. Petiton and Lamine M. Aouad: Large Scale Peer to Peer Performance Evaluations, with Gauss-Jordan Method as an Example. *HeteroPar'03, in conjuction with PPAM 2003*.

[16] Lamine M. Aouad and Serge G. Petiton: Experimentation and Programming Paradigms for Matrix Computing on Peer to Peer Grid. GRID 2004, the *fifth IEEE/ACM International Workshop on Grid Computing.*

[17] Franck Cappello, Samir Djilali, Gilles Fedak, Thomas Hérault, Frédéric Magniette, Vincent Néri: Computing on large-scale distributed systems: XtremWeb architecture, programming models, security, tests and convergence with grid. *Future Generation Comp. Syst*. 21(3): 417-437 (2005)

[18] Lamine M. Aouad: Contribution a l'algorithmique matricielle et evaluation de performances sur les grilles de calcul vers un modele de programmation a grand echelle. *PhD thesis.*

[19] Olivier Delannoy and Nahid Emad: YML/LAKeWeb Portal: Web based research portal for linear algebra application. HPC 2006: *High Performance Computing Symposium, April 2006 Huntsville AL, USA.*

[20] http://yml.prism.uvsq.fr/

[21] http://kadeploy.imag.fr/

**Ling Shang** is a PhD candidate of Lifl at USTL. He is a member of MAP team of Lifl and Grand Large Team of INRIA Futurs in France. His interests include high performance computing, Grid computing, Cloud computing and numerical algorithm on internet-based computing environments.

**Maxime Hugues** is a PhD candidate of Lifl at USTL. He is a member of MAP team of Lifl and Grand Large Team of INRIA Futurs in France. He also works for TOTAL. His interests include high performance computing, Grid computing, Multi Core and numerical algorithm on internet based computing environments

**Serge G. Petiton** received his Ph.D. degree in 1988. Prof. Petiton leads the "Methodology and Algorithmic Programming" group of the CNRS "Laboratoire d.Informatique Fondamentale de Lille", and he actively participates in the INRIA "Grand Large" project. He is director of the board of the ORAP Association to promote HPC and participates in several French, European and International HPC committees. Currently his main research interests are in Parallel and Distributed Computing, Multi-Core and Hybrid Computing, High Performance Linear Algebra, and Grid and Cloud Computing.