2009 Eighth International Conference on Grid and Cooperative Computing

A New Parallel Paradigm for Block-based Gauss-Jordan Algorithm

Ling Shang, Serge Petiton and Maxime Hugues

LIFL, University of Science and Technology of Lille
Grand-Large Team, INRIA Futurs
Lille, France
(ling.shang, serge.petiton, maxime.hugues)@lifl.fr

***Abstract*—Two kinds of parallel possibilities which are intra-step and inter-steps parallelism exist in the block-based Gauss-Jordan algorithm which is a classical method of large scale matrix inversion. But the existing parallel paradigm of Block-based Gauss-Jordan algorithm just aiming at the intra-step parallelism, can't meet the requirement of making more tasks executed simultaneously in high performance platform can be harnessed more and more computing resources. To overcome the problem described above, this paper presents a hybrid parallel paradigm exploiting all the possible parallelizable parts of the Gauss-Jordan algorithm. In this hybrid parallel paradigm, 1) Divide and conquer paradigm is responsible for decomposing the large granularity task into sub-tasks as much as possible; 2) Single program multi data (SPMD) paradigm deals with intra-step parallelism in the algorithm; 3) Data pipelining paradigm helps to solve the problem of inter-steps parallelism. Finally some experiments based on comparison the hybrid parallel paradigm with the existing parallel paradigm show us the good performance of our paradigm.**

***Keywords*-Gauss-Jordan algorithm; parallel paradigm; data dependence, parallelism**

I. Introduction

A good parallel programming paradigm should help to maximize parallel execution of the algorithm, thus achieving better performance. And the choice of paradigm is determined by the available parallel computing resources and by the type of parallelism inherent in the problem [14].

Exploiting the significant computational capability available in the internet-based Grid environment has gained an enthusiastic acceptance within the high performance computing community, and the current tendency favors this sort of commodity supercomputing [10]. Multi-core Architectures (MCAs) provide applications with an opportunity to achieve much higher performance and the number of cores on MCAs is likely to continue growing, increasing the performance potential of MCAs [12]. All those technologies are mainly motivated by the fact that most of the scientific communities have the desire to minimize economic risk and rely on consumer based off-the-shelf technology. Grid computing and multi-core have been recognized as the wave of the future to solve large scientific problems. However, realizing this performance potential in an application requires the application to expose a significant amount of thread-level parallelism. It is important to find a solution to get maximal parallelism in a certain algorithm for researchers, thus exploiting computing resources in the Grid platform or MCAs as much as possible.

Block-based Gauss-Jordan algorithm [1][2][4], as a classical method of large scale matrix inversion, can be used in weather prediction, aircraft design, graphic transformation and so on. Its high availability in many domains makes it become the focus of many researchers. Serge shows the parallel version of the algorithm adapting to MIMD [1]. N. Melab et al not only give us its parallel version tailoring to MARS but also analyze all the possible parallelism in the algorithm [2][4]. L. M. Aouad et al present its parallel programming paradigm based on SPMD [7]. As well known to us all, paradigm is a class of algorithms that have the same control structure and we can very easily tailor it to any execution models such as MPI, PVM and other middleware suiting for high performance computing. And a good programming paradigm is very important for an algorithm to get better performance. But programming paradigm given by N. Melab and L. M. Aouad doesn't take inter-steps parallelism into account. To improve the efficiency of the algorithm, it is important and necessary to find a solution which can exploit all the inter-steps and intra-step parallelism in the algorithm, thus generating more tasks and making these tasks executed simultaneously. So analysis based on the sequential block-based Gauss-Jordan algorithm has been made and some characters can be summarized as follows: 1) All the objects of operation are data blocks and the sequence of operations in the algorithm is decided by data dependence between different blocks; 2) The number of steps of algorithm execution is equal to the number of data-blocks divided into; 3) the parallelism of basic operation in the algorithm depends on the data write-operation ; 4) the number of data write-operation is same in each iterative step. This analysis can tell us that data dependence between different blocks in the algorithm plays a very important role. So this paper emphasizes on the analysis of data dependence of different blocks and table is used to simulate the real matrix manipulation. Then formal description based on table simulation is made to demonstrate the existed data dependence between different blocks. At the same time, write-operation which controls the data dependence

978-0-7695-3766-5/09 $25.00 © 2009 IEEE
DOI 10.1109/GCC.2009.75

on different data blocks in the algorithm is another key factor. So a global flag is created to control the data write-operation during the process of algorithm execution. Thus, all the data dependence in the algorithm can be dealt with using these flags. Based on that, this paper presents a hybrid programming paradigm in which: 1) divide and conquer paradigm guides us to choose appropriate block sizes; 2) Data pipelining paradigm is used to make inter-steps based parallel parts in the algorithm executed in parallel; 3) SPMD paradigm works on the intra-step parallelism.

The remainder of the paper is organized as follows: the second section introduces related work about the research. The data dependence in the algorithm will be described in the section three. The fourth section will present us a hybrid parallel programming paradigm. The evaluation of hybrid paradigm will be made in section five. Section six is the conclusion and future works.

II. Data Dependence in the Algorithm

Let *A* and *B* be two dense matrices of dimension N, and let *B* be the inverted matrix 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. The sequential block-based Gauss-Jordan algorithm can be presented as follows:

Algorithm 1

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

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

for k = 1 to q do

$A^k_{k,k} \leftarrow (A^k_{k-1,k})^{-1}$ **(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

A. Parallelism in the Algorithm

From the analysis of data dependence of the algorithm 1, we can find that each step of algorithm is made up of five loops (the third, fourth, fifth, sixth and seventh operation in algorithm 1) and two other operations (the first and second). At each step k (k = 1,, q), the first operation is used to get the inverted matrices of sub-block matrices; the second operation executes the operation of assignment from the sub-block of matrix A to corresponding that of matrix B; the third operation computes the blocks belonging to the row of the pivot with index j is larger than k and the fourth operation computes the corresponding parts of matrix B; the fifth operation calculates the blocks of all columns of the matrix A with index i above and below that of the pivot row and the forth operation calculates the corresponding parts of matrix B; At last the seventh operation is used to compute the blocks of the column number k of matrix B except Bk,k.

The parallelization of the sequential block-based G-J algorithm consists of exploiting two kinds of parallelism: the inter-steps parallelism and the intra-step parallelism. The intra-step parallelism consists mainly of exploiting the parallelism involved in each of the five loops (operation '3' to operation '7' in algorithm 1). It falls into two categories: the inter-loops parallelism and the intra-loop parallelism. Inter-steps parallelism is more complex because almost all the operations in algorithm 1 are restricted by the inter-steps data dependence. We will analyze them in the section 2.3 of this part.

Write operation in each block is executed only once in one step. These operations can be divided into two parts according to the goal (matrix A or matrix B) will be written into. Operations '1', '3' and '5' are on matrix A, while operation '2', '4', '6' and '7' are on matrix B. So we can analyze the data dependence according to operations made on matrix A or matrix B. All these data dependence which rest on the parallelism of algorithm will be analyzed one by one.

1) Intra-step Parallelism in the Algorithm: The inter-loops parallelism means that the loops are executed in parallel, the data dependence between the loops must be managed. In the sequential algorithm, we identify three ones: 1) the internal loop of the second, third, fourth and seventh operation(see the number marked in algorithm 1) can't be executed before we get the value of block $A^k_{k,k}$; 2) the internal loop of the fifth operation depends on the third one because it uses the block $A^k_{k,j}$ which is computed by the third operation; 3) the internal loop of the sixth operation is dependent on the fourth operation because it uses the block $B^k_{k,j}$ which is computed by the fourth operation.

2) Inter-steps Parallelism in the Algorithm: The intra-loop parallelism consists of executing each of the five loops of the algorithm in parallel. Particularly, the internal loops of the fifth and sixth operation are run each of them in parallel. The work units for these loops are of two kinds: matrix

product and matrix triadic respectively $X \times Y$ and $Z - X \times Y$where *X*, *Y* and *Z* are matrix blocks. Moreover, a crucial criterion which must be taken into account is the granularity of the work units. Indeed, the size of the matrix blocks involved into each work unit must be such that its computation cost balances the communication cost caused by sending tasks to the remote node. The experiment part of this paper will show us the influence on the efficiency of algorithm caused by changing the sub-block's size and count of matrix.

B. Inter-step Based Parallelism

Though different operations on matrix *A* and matrix *B* existing, the method of operation on them is similar. So we can analyze those data dependence decided by operational sequence together.

We will take the data-dependence between step *k* and step $k+1$ as example. 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 of block $A_{i,j}$ $(i = 1, ..., q \quad and \quad i \neq k; j = k+1, ..., q)$ and $A_{i,k}(i = 1, ..., q \quad and \quad i \neq k; k = 1, ..., q)$ of k step before we compute the value $A_{i,j}(i = 1, ..., q \quad and \quad i \neq k; j = k+1, ..., q)$ of $k+1$ step. The situation in the sixth operation is just like that in the fifth, we can compute the value $B_{i,j}(i = 1, ..., q \quad and \quad i \neq k; j = 1, ..., k-1)$ of step $k+1$ after we get the value $B_{i,j}(i = 1, ..., q \quad and \quad i \neq k; j = 1, ..., k-1)$ and $A_{i,k}(i = 1, ..., q \quad and \quad i \neq k; k = 1, ..., q)$ of step k. In the operation '7', to get the value $B_{i,k}(i = 1, ..., q \quad and \quad i \neq q; k = 1, ..., q)$ of $k+1$ we must know the value $A_{i,k}(i = 1, ..., q \quad and \quad i \neq q; k = 1, ..., q)$ of step k.

To express the inter-steps data dependence, a data dependent graph will be drawn. First, we define "directed arc" as follows: a directed arc from domain block "X" to domain block "Y" (X, Y is the block of matrix) signifies the operation on block "Y" should not be executed before the operation on block "X" is finished. See directed arc in Fig.2. 'Loop 1' in the Fig.2 represents the inter-step data dependence of operation '1' and operation '5'. Operation '5' at the step 2 must finish before we execute the operation '1' at the step 3. Fig 2 shows us all the inter-steps data dependence existing between step 2 and step 3 with q is 5.

Then we can summarize all the data dependence as follows (see Fig 3):

The letter k and k+1 stand for the step k and step k+1 during algorithm execution. The number of behind letter is used to show the operations marked in algorithm 1.

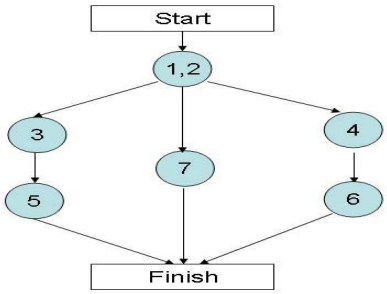

Figure 1. Intra-step Data Dependence

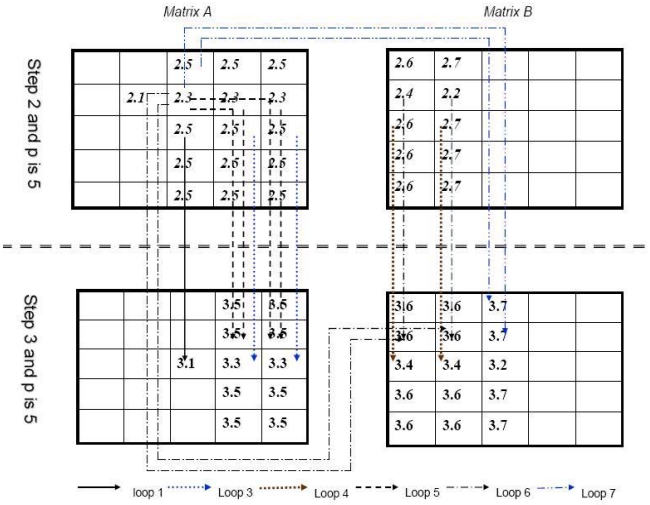

Figure 2. Inter-steps Data Dependence

III. Formal Description of Data Dependence

It is the regularity of operation on matrix A and matrix B that remind us to take matrix A and matrix B together into account as an augmented matrix.

Definition 1: Let

$AB(i,j) = A(i,j) \quad i = 1, ..., q; j = 1, ..., q$

$AB(i,j+q) = B(i,j) \quad i = 1, ..., q; j = 1, ..., q$

$AB^k_{i,j}$ represents the block $AB_{i,j}$ at step *k* for $k = 1, ..., q$ For the associated algorithm one $AB^k_{i,j}$ is modified once and only once per step, Let us represent domain $AB^k_{i,j}$ using a triplet *(k, i, j)*, Then we can use the three triplet $KIJ = (k,i,j)|k,i = 1, ..., q \quad and \quad j = k+1, ..., k+q$ to mark the status of block operation, i.e. the operation is made at the step *k* in the row *i* and column *j* of augmented matrix $AB_{i,j}$. For the algorithm, one AB_{ki}, j is modified once and only once per step. So these triplets can be used to describe the sequence of operation on augmented matrix. Consequently we can use the triplet as a global signal to control the data write operation, thus the data dependence in the algorithm can be describe using these triplets.

Definition 2: Let the binary relation $\Longrightarrow$ be defined as

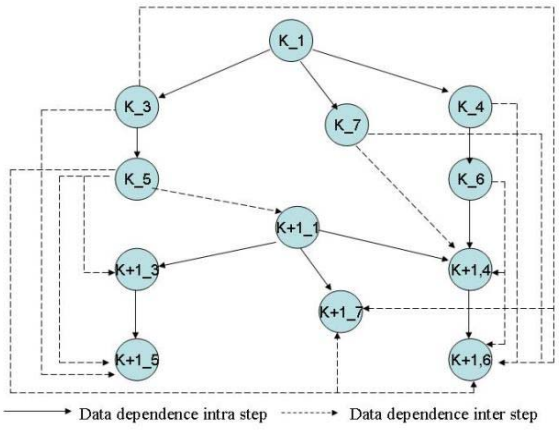

Figure 3. All the Data Dependence in the algorithm.

follows: $X \Longrightarrow Y$ (*X, Y* is the block of matrix) if and only if there exists an edge from *X* to *Y* in the Fig 3. The data dependence can thus be represented by the triplets defined above. For all the data dependence existing in Fig 3, we have:

Intra-step data dependence can be written as follows:

- (k, k, k) $\Longrightarrow$ (k, k, j) $k = 1, \ldots, q; j = k+1, \ldots, q$
- (k, k, k) $\Longrightarrow$ (k, k, j+q) $k = 1, \ldots, q; j = 1, \ldots, k-1$
- (k, k, j) $\Longrightarrow$ (k, i, j) $k = 1, \ldots, q; j = k+1, \ldots, q; i = 1, \ldots, q \quad and \quad i \neq k$
- (k, k, j+q) $\Longrightarrow$ (k, i, j+q) $k = 1, \ldots, q; j = 1, \ldots, k-1; i = 1, \ldots, q \quad and \quad i \neq k$
- (k, k, k) $\Longrightarrow$ (k, i, k+q) $k = 1, \ldots, q; i = 1, \ldots, q \quad and \quad i \neq k$

Inter-step data dependence can be summarized as follows:

- (k-1, k, k) $\Longrightarrow$ (k, k, k) $k = 2, \ldots, q$
- (k-1, k, j) $\Longrightarrow$(k, k, j) $k = 2, \ldots, q; j = k+1, \ldots, q$
- (k-1, k, j+q) $\Longrightarrow$ (k, k, j+p) $k = 2, \ldots, q; j = 1, \ldots, k-1$
- (k-1, i, k) $\Longrightarrow$ (k, i, j) $k = 2, \ldots, q; j = k+1, \ldots, q; i = 1, \ldots, q \quad and \quad i \neq k$
- (k-1, i, j) $\Longrightarrow$ (k,i, j) $k = 2, \ldots, q; j = k+1, \ldots, q; i = 1, \ldots, q \quad and \quad i \neq k$
- (k-1, i, j+q) $\Longrightarrow$ (k, i, j+p) $k = 2, \ldots, q; j = 1, \ldots, k-1; i = 1, \ldots, q \quad and \quad i \neq k$
- (k-1, i, k) $\Longrightarrow$ (k, i, j+p) $k = 2, \ldots, q; j = 1, \ldots, k-1; i = 1, \ldots, q \quad and \quad i \neq k$
- (k-1, i, k) $\Longrightarrow$ (k, i, k+p) $k = 2, \ldots, p; i = 1, \ldots, p \quad and \quad i \neq k$

Now we will summarize binary relations obtained above according to operations marked in algorithm 1.

- (k, k, k) $\Longrightarrow$ (k, k, j) $k = 1, \ldots, q; j = k+1, \ldots, q$ **(1)**
- (k, k, k) $\Longrightarrow$ (k, k, j) and (k-1, k, j) $\Longrightarrow$ (k, k, j) $k = 2, \ldots, q; j = k+1, \ldots, q$ **(2)**
- (k, k, k) $\Longrightarrow$ (k, k, j+q) and (k-1, k, j+q) $\Longrightarrow$ (k, k, j+q) $k = 2, \ldots, q; j = 1, \ldots, k-1$ **(3)**
- (k, k, j) $\Longrightarrow$ (k, i, j) and (k-1, i, k) $\Longrightarrow$ (k, i, j) and (k-1, i, j) $\Longrightarrow$ (k,i, j) $k = 2, \ldots, q; j = k+1, \ldots, q; i = 1, \ldots, q \quad and \quad i \neq k$ **(4)**
- (k, k, j+q) $\Longrightarrow$ (k, i, j+q) and (k-1, i, j+q) $\Longrightarrow$ (k, i, j+q) and (k-1, i, k) $\Longrightarrow$ (k, i, j+q) $k = 2, \ldots, q; j = 1, \ldots, k-1; i = 1, \ldots, q \quad and \quad i \neq k$ **(5)**
- (k, k, k) $\Longrightarrow$ (k, i, k+q) and (k-1, i, k) $\Longrightarrow$ (k, i, k+q) $k = 2, \ldots, p; i = 1, \ldots, p \quad and \quad i \neq k$ **(6)**
- (k, k, k) $\Longrightarrow$ (k, i, k+q) and (k-1, i, k) $\Longrightarrow$ (k, i, k+q) $k = 2, \ldots, p; i = 1, \ldots, p \quad and \quad i \neq k$ **(7)**

IV. Hybrid Parallel Programming Paradigm

There is no simple recipe for designing parallel algorithms. However, it can benefit from a methodological approach that maximizes the range of options, that provides mechanisms for evaluating alternatives, and that reduces the cost of backtracking from wrong choices [14]. The design methodology allows the programmer to focus on machine-independent issues such as concurrency in the early stage of design process, and machine-specific aspects of design are delayed until late in the design process. As suggested by Ian Foster [13], this methodology organizes the design process into four distinct stages which are: partitioning, communication, agglomeration and mapping. Next we will design the parallel paradigm of Gauss-Jordan algorithm according to those four steps presented above.

Because large scale matrix inversion is hard to be executed directly, so partitioning is the first thing to do. In this paper, large scale Matrix A has been partitioned into n×n blocks. After partitioning, the objects of all the operations in the algorithm focus on sub-matrices, which makes a more complex problem to be decomposed into some easy-to-solve sub-problems. No communication between different small sub-problems exists during the process of program execution. The communication is just between matrix A and its sub-matrix. The operation on sub-matrix can read from (or write back to) the corresponding block of Matrix A. We can design this stage of paradigm according to parallel divide and conquer model in which the sub-problems can be solved at the same time, giving sufficient parallelism.

Analysis in the Section 2 tells us that inter-steps and intra-step parallelism exist in this algorithm and these parallelisms rest on the data dependence. In this algorithm, data dependence on different blocks determine the sequence of different operations on a sub-matrix. As to inter-step data dependence in Matrix A, there are (n-1)×(n-k)blocks(k is the number of steps)which have the same write-operation (operation '5'). They have the same operation and sequence

of that and there is no communication between them. This means that the sub-tasks of the algorithm are independent and each processor can execute one part of them. So this kind of operations can use single program multi-data paradigm model to deal with. As to Fig 2, the operation represented by loop '1' shows us that before the execution of operation '1' in step 3 begins, the operation '5' in step 2 must finish. This kind of parallelism we can use data pipelining paradigm to deal with them.

All the write-operations of the algorithm are based on sub-matrix and the results will be returned to the Matrix A. So when all the operations on sub-matrix are end, agglomeration is finished. Hybrid parallel paradigm can generate sub-tasks as many as possible, So you can use any kind of schedule stratagem to map the sub-tasks to computing resources.

Next, a Flow-chart of hybrid programming paradigm can be see in Fig 4. Based on the analysis above, a formal description of parallel paradigm of block-based Gauss-Jordan algorithm can be presented as follows:

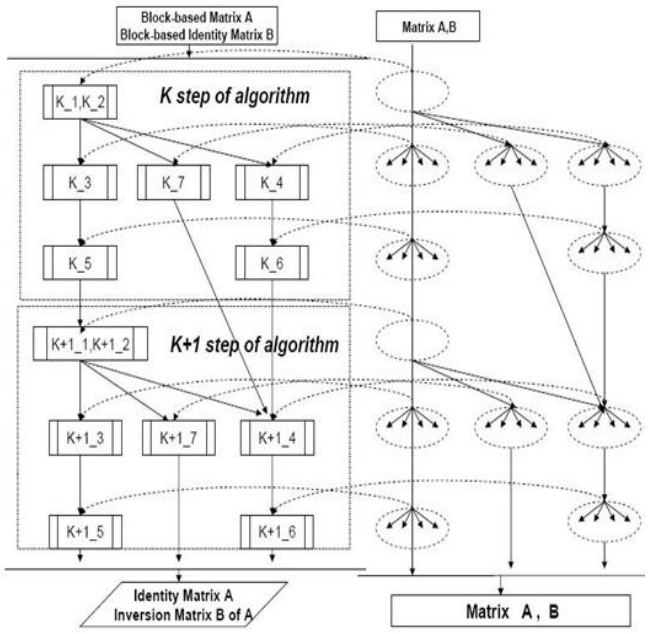

Figure 4. Left part is the flowcharts of the paradigm execution and the right part is the corresponding data block operation. Q steps needed to execute in the paradigm and the execution between step k and step k+1 is described. In the right parts, many arrows represents these data can be executed simultaneously..

According to formal description, a set of three-touples as the global signals are used to control the execution of algorithm. Through the Fig 4, we can know the flowchart of algorithm execution. Then the hybrid parallel programming paradigm can be presented as follows: (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 on the basis of their conditions are needed.)

New Parallel Paradigm for 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

V. Performance Evaluation

In this section, we first present the influence of the granularity of parallelism on the efficiency of the parallel execution of the algorithm. Then, we study the scalability of the algorithm. All this experiments are based on YML/OmniRPC [5][6][9] (middleware) and Grid5000 [8] (platform, France) through the comparison the hybrid paradigm with intra-step based parallel programming paradigm. Here what we want to emphasize is that all the experiments are just to show the performance of hybrid programming paradigm, no special middleware or environments is necessary, you can choose any what you like to use. In our experiment, the computational resources can be described as follows:

Table I
Resources in Grid'5000

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

A. Granularity of parallelism

Experiment motivation: test the performance of hybrid paradigm with different granularity. Two methods to change the granularity of parallelism in this experiment are changing block-size with block-count fixed and changing the block-count with block-size fixed.

Experiment environment: First, 8 nodes × 2=16 cores are used in the cluster Bordereau, Bordeaux site, France. Second 100 nodes × 2=200 cores are used in the cluster grelon, Nancy site, France.

1) block-size changed and block-count fixed: From the Fig 5, we can find that when the block-size is not large (block-size is 50 in this experiment), the elapsed time of algorithm based on hybrid paradigm is longer than that based on intra-step paradigm. Then time consumed of hybrid paradigm becomes shorter than that of intra-step paradigm with the increase of block-size. But the difference between them is not large when the block-size is less than 500. After that, time difference becomes larger. The reason is that when block-size is small, time consumed in computing is short and the advantage of more tasks can be executed in parallel in the hybrid paradigm is not made full use of. In this situation, communication and schedule time take a great proportion in the overall time. The reason is that more tasks can be generated and executed in parallel in hybrid paradigm makes the overhead is larger than that in intra-step paradigm. But in those scenarios which computation time plays a dominant role, the advantage of hybrid paradigm is obvious (see the figure 5 when the block-size is larger than 500).

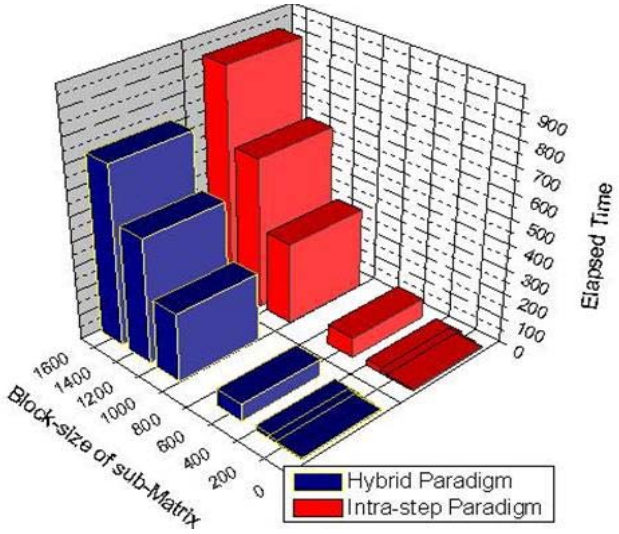

Figure 5. Comparison of elapsed time with block-size changing

Here what I want to emphasize is that in the ideal condition, the performance of hybrid paradigm is better than that of intra-step based parallel paradigm in any case. We can get this conclusion from the analysis of data dependence between different blocks. In real execution environments, overhead of YML plays an important role in the block-size is small. Maxime et al evaluate the overhead of YML in his paper [15].

2) block-count changed and block-size fixed: In this part, we will analyze the change of time difference with the condition of block-count change. Table 2 tells us that either the increase of block-size or the increase of block-count all makes the time difference become longer. Increase amplitude of time difference becomes larger with the bigger of block-size. The results are reasonable, because with the increase of block-size, the computing time takes a great proportion and better concurrency in the hybrid paradigm ensures maximizing parallelism of program execution, which shorten the wait time comparing with intra-step paradigm. The same reason can explain the gap of time difference becomes larger with the increase of block-count. In the program based on hybrid paradigm, less wait time caused by data dependence is consumed in the process of program execution. Because hybrid paradigm make all the possible parallelizable parts executed in parallel and intra-step based paradigm just ensure the parallelism in the loop without considering the inter-step parallelism. More importantly, when the block-count is 8 and block-size is 1500, the time difference is more than 1880s. This is a very large gap and with the increase of block-count and block-size, the gap will be larger. The advantage of hybrid parallel paradigm becomes more obvious.

Table II
TIME DIFFERENCE WITH THE CHANGE OF BLOCK-COUNT AND BLOCK-SIZE

Block-count	**Block-size**	**Block-size**	**Block-size**
	500×500	**1000×1000**	**1500×1500**
2 × 2	1.76	4.73	12.91
3 × 3	3.18	16.73	94.49
4 × 4	11.08	53.41	165.02
5 × 5	20.96	101.36	466.72
6 × 6	34.58	192.4	707.4
7 × 7	40.21	298.6	1082.4
8 × 8	51.58	430.5	1885.23

B. Scalability

Experiment motivation: Test the scalability of program based on hybrid paradigm. The method is that we will execute the same program in the same environment, but different computing resources.

Experiment Environment: cluster grelon, Nancy site, France; 10 nodes×2=20 cores; 100 nodes×2=200 cores. The reason for using 20 cores and 200 cores is that 20 cores can not meet the requirement of all tasks parallelism while 200 cores can.

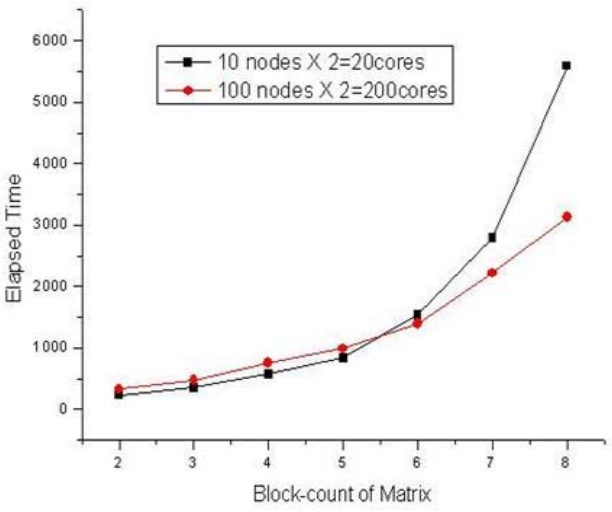

Figure 6. Elapsed time with different computing resources to compute the same program in the same environment

Fig.6 shows us that when the block-counts is less than 5×5, the elapsed time using 20 cores is less than that using 200 cores. The reason is that the schedule of YML needs more time to solve the data dependence in the environments with 200 cores and less data between the data server and computing nodes is delivered, thus saving a lot of computing time [3]. The accidence comes from the middleware we use and it is not the problem of parallel paradigm (see overhead of YML in paper [15]). In the ideal scenario, the elapsed time with the 20 cores and that with 200 cores is almost the same. But when the block-count is larger than 5×5, 20 cores can not meet the requirement of more computing resources with the increase of sub-tasks generated, while 200 cores can, So the advantage of using 200 cores becomes more obvious, showing us that the program has a good performance in scalability under the condition of without considering the influence of real running environment.

VI. CONCLUSION AND FUTURE WORK

This paper presents a new parallel paradigm based on the analysis of data dependence in the algorithm. The paradigm, which exploits all the possible parallelizable parts of the algorithm, can help us improve the performance of block-based Gauss-Jordan algorithm on the Grid platform or Multi-core Architectures in which large amount of computing resources have been harnessed. The experiment also testify the good performance when solving a real problem, and its scalability makes sure the paradigm can be tailored to any high performance platform very easily.

But there is still something puzzling to us what is the relation of the number of concurrency tasks and the number of computing nodes/cores. Another problem is that the time complexity of this paradigm needs to be measured.

ACKNOWLEDGMENT

The authors would like to thank CROUS Lille and CNOUS for their support. In addition, we thank a lot Olivier Delannoy and Nahid Emad from PRiSM, Laurent Choy and Mitsuhisa Sato from the University of Tsukuba for their help and collaboration on YML and OmniRPC. We thank a lot INRIA for their help on Grid5000.

REFERENCES

[1] S. Petiton: Parallelization on an MIMD computer with realtime Scheduler. Aspects of Computation on Asynchronous Parallel Processors, North Holland, (1989).

[2] N. Melab, E.-G. Talbi, and S. G. Petiton: A parallel adaptive Gauss-Jordan algorithm. The Journal of Supercomputing, 17(2):167C185, (2000).

[3] M. Hugues. Algorithmique scientifique distribu grande chelle et programmation yml sur cluster de clusters. mastersthesis, (2007).

[4] N. Melab, E.-G. Talbi, and S. G. Petiton: A parallel adaptive version of the block-based gauss-jordan algorithm. In IPPS/SPDP, pages 350C354, (1999).

[5] 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 362C 369. IEEE Computer Society Press, (2004).

[6] O. Delannoy, N. Emad, and S. G. Petiton. Workflow Global Computing with YML. In The 7th IEEE/ACM International Conference on Grid Computing, pages 25C32, (2006).

[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 et al: Grid5000: a large scale and highly reconfigurable grid experimental testbed. In The 6th IEEE/ACM International Conference on Grid Computing, pages 99C106, (2005).

[9] M. Sato, T. Boku, and D. Takahashi. OmniRPC: a Grid RPC ystem for Parallel Programming in Cluster and Grid Environment. In The 3rd IEEE International Symposium on Cluster Computing and the Grid, pages 206C, (2003).

[10] S. T. Ian Foster, C. Kesselman. The Grid: Blueprint for a Future Computing Infrastructure. Morgan Kaufmann, USA, (1999).

[11] J. M. Anderson and M. S. Lam. Global optimizations for parallelism and locality on scalable parallel machines. In Proceedings of the ACM SIGPLAN 93 Conference on Programming Language Design and Implementation, pages 112C125, Albuquerque, New Mexico, June 23C25, 1993. SIGPLAN Notices, 28(6), June (1993).

[12] Miquel Perics, Ruben Gonzalez, Francisco J. Cazorla, Adrian Cristal, Daniel A. Jimenez and Mateo Valero.: A Flexible Heterogeneous Multi-Core Architecture. Parallel Architecture and Compiler Techniques (PACT-2007)

[13] Designing Parallel Algorithms : http://www-unix.mcs.anl.gov/dbpp/text/book.html

[14] Parallel Programming Models and paradigms: http://www.buyya.com/cluster/v2chap1.pdf

[15] Maxime Hugues and Serge G. Petiton: A Matrix Inversion Method with YML/OmniRPC on a Large Scale Platform. VECPAR'2008, Page: 95-108 Jun 24-27, 2008, Toulouse, France