DOI: 10.2298/CSIS100907026S

Parallel Processing on Block-based Gauss-Jordan Algorithm for Desktop Grid

Yizi SHANG[1], Guiming LU[2], Ling SHANG[2,3], and Guangqian WANG[1]

[1] State Key Laboratory of Hydro-science and Engineering, Tsinghua University, Beijing, China
syz05@mails.tsinghua.edu.cn
[2] North China University of Water Conservancy and Electric Power, School of Information Engineering, Zhengzhou, China
njshangl@gmail.com
[3] Lifl, University of Science and Technology of Lille, Lille, France
ling.shang@lifl.fr

Abstract. Two kinds of parallel possibilities exist in the block-based Gauss-Jordan (BbGJ) algorithm, which are intra-step and inter-steps based parallelism. But the existing parallel paradigm of BbGJ algorithm just aiming at the intra-step based parallelism, can't meet the requirement of dispatching simultaneously as many tasks as possible to computing nodes of desktop grid platform exploiting thousands of volunteer computing resources. To overcome the problem described above, this paper presents a hybrid parallel paradigm for desktop grid platform, exploiting all the possible parallelizable parts of the BbGJ algorithm. As well known to us all, volatility is the key issue of desktop grid platform and faults are unavoidable during the process of program execution. So the adapted version of block BbGJ algorithm for desktop grid platform should take the volatility into consideration. To solve the problem presented above, the paper adopts multi-copy distribution strategy and multi-queue based task preemption method to ensure the key tasks can be executed on time, thus ensure the whole tasks can be finished in shorter period of time.

Keywords: Gauss-Jordan algorithm, desktop grid, data dependence, parallelism, hybrid parallel paradigm.

1. 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].

Most of the scientific communities have the desire to minimize economic risk and rely on consumer based off-the-shelf technology. Exploiting the significant computational capability available in the internet-based desktop

Grid environment has gained an enthusiastic acceptance within the high performance computing community [10,18] and Desktop Grid computing has been recognized as the wave of the future to solve large scientific problems, especially in the days of multi-core architectures [12] are widely used in personal computer. Paper [19] evaluates the potential capability of desktop grid systems. With more and more personal computers with multi-core architectures join the desktop grid platform, the platform can provide huge process However, realizing better performance of the parallel algorithm in desktop grid platform requires exploiting as more thread-level parallelism as possible and avoiding the influence from computing nodes joining in (leaving from) the platform without any advance notice. It is important to find a solution to get maximal parallelism available to an algorithm and find appropriate schedule mechanism for the paltform, thus achieving its better performance in the desktop Grid platform.

BbGJ algorithm [1][2][4], as a classical method of large scale matrix inversion, can be used in hydro-science, weather prediction, aircraft design, graphic transformation and so on. Though there are many approaches on large scale matrix inversion such as SCALAPACK library, GOTO library. But these kinds of methods aren’t available to desktop grid paradigm. The reason is that it is impossible to install a special library on each computing nodes. It is meaningful to make further research on BbGJ algorithm. Serge shows its parallel version adapting to MIMD [1]. 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]. Aouad et al present its adapted version for grid platform [3, 7]. But these programming paradigm given by Melab and Aouad doesn’t take inter-steps parallelism into consideration. More important, they can’t be adapted to desktop grid environment for the special characters of desktop grid systems.

To improve the efficiency of the algorithm, it is important and necessary to find an approach which can exploit all the inter-steps and intra-step based parallelism in the algorithm. Thus more tasks can be generated and executed simultaneously on different computing nodes. Some characters of sequential BbGJ can be summarized as follows:

1) The number of iterative step of sequential BbGJ algorithm is equal to that of square root of matrix blocks partitioned.

2) In the BbGJ algorithm, the parts of available parallel granularity are based on blocks (which are sub-matrices partitioned). data dependence between blocks plays an important role in deciding which parts of program can be executed concurrently.

3) All the data input and output are based on blocks and the number of data input is same in each iterative step of BbGJ algorithm.

4) The object of data input on blocks is regular. Operation ’1’,’3’,’5’ (marked in sequential BbGJ algorithm in section 2) work on the matrix A while Operation ’4’,’6’,’7’ work on the matrix B which is the inversion of matrix A (see Fig.6).

From above, we know data dependence between different blocks is the key factor to solve the problem presented above. To the convenient of making research on data dependence between blocks, we regard the two matrix (original matrix and its inverted matrix) as an augmented matrix. Through the analysis on executive process of BbGJ algorithm, we find that a set of three-tuples (k,i,j) can be used to control all the data dependence in the process of algorithm execution. In which, k represents the number of iterative step of BbGJ algorithm, i represents the row of augmented matrix while j is the column of the matrix. So those three-tuples are used as condition to control when and which parts can be executed. At the same time, we can find and mark the key path of depended tasks and the key path can be described using a series of three tuples. Based on that, this paper presents a hybrid programming paradigm, considering all the possible parallelism in the algorithm and appropriate allocation strategy.

The remainder of the paper is organized as follows: The data dependence in BbGJ will be described in the section two. Section three will show us the formal description of BbGJ The fourth section will present us a hybrid parallel programming paradigm for desktop grid system and key path of BbGJ using a series of three-tuples. The evaluation on hybrid paradigm will be made in section five. Section six is the conclusion and future works.

2. 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 is the following:

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

For k=1 to q do                          // we call this as "iterative"
        $A^{k}_{k,k} \leftarrow (A^{k-1}_{k,k})^{-1}$                                   (1)
        $B^{k}_{k,k} \leftarrow A^{k}_{k,k}$                                           (2)
    For j=k+1 to q do                    // we call this as "loop"
        $A^{k}_{k,j} \leftarrow A^{k}_{k,k} A^{k-1}_{k,j}$                             (3)
    End For
    For j=1 to k-1 do                    // we call this as "loop"
        $B^{k}_{k,j} \leftarrow A^{k}_{k,k} B^{k-1}_{k,j}$                             (4)
    End For
    For j=k+1 to q do                    // we call this as "loop"
        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              // we call this as "loop"
           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       // we call this as "loop"
           $B^k_{i,k} \leftarrow - A^{k-1}_{i,k} A^k_{k,k}$                       (7)
        End For
 End For
```

2.1. Parallelism in the Algorithm

From 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 $B_{k,k}$.

The parallelization of the sequential BbGJ 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. Then we will analyze those data dependence one by one.

2.2. Intra-step based data dependence in the Algorithm

We can describe the data dependence in the Algorithm1 using Fig.1 and Fig.2. In those figures the blue font represents the read operation. For example, the blue font '5' in the first row, second column of matrix *A* in Fig.1 represents that: to this block, the first operation is '5' (see the number marked in algorithm 1) which is a read operation, i.e. operation '5' will read this block as its input. The same way we can know that the black font '1' in the second row, second column of matrix *A* in Fig 1 represents that: the operation '1' (see the number marked in algorithm 1) will assign the value obtained by operation '1' to this block. Next we will take the block in the second row, first column of matrix *B* in Fig.1 as example to show all the related operations executed in this block.

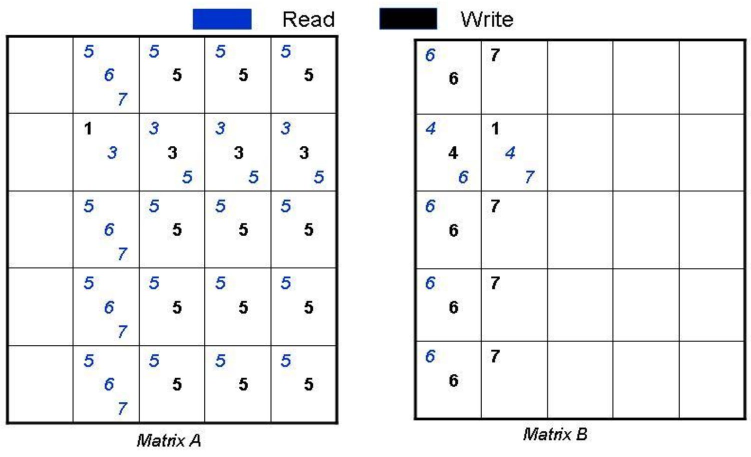


Fig.1. Blocks Operated at the Step 2 with q is 5 in Algorithm 1

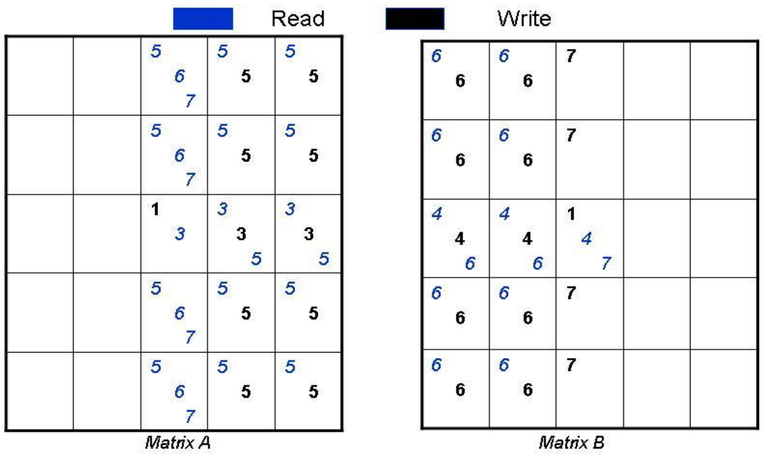


Fig.2. Blocks Operated at the Step 3 with q is 5 in Algorithm 1

From the Fig.1, we can know three operations executed in this block, of which blue fonts ‘4’ and ‘6’ are read operation and black font ‘4’ is write operation. How those operations are executed and what’s the sequence of those operations will be explained. To that block said above, operation will be executed from up to down, i.e. the read operation ‘4’ will be executed first, after that write operation ‘4’ begin to execute and the last operation is read operation ‘6’. After the finish of the three operations said above, no more operation on this block will be executed in this step. Fig.1 shows us all the operations happened at the step 2 with *q* is 5 in the intra-step BbGJ algorithm. We can also get all the operations information at the step 3 with *q* is 5 from the Fig.2.

From Fig.1 and Fig.2 we can know that each of the q steps of the algorithm has the same number of write operation and the number is q^2. To the matrix *A* ,the write operation on it are operations '1', '3' and '5' while only operations '1', '6' and '7' act on matrix *B*. Through the analysis of Fig.1 and Fig.2, we also can get the data dependence in an iterative step of program execution. See Fig.3.

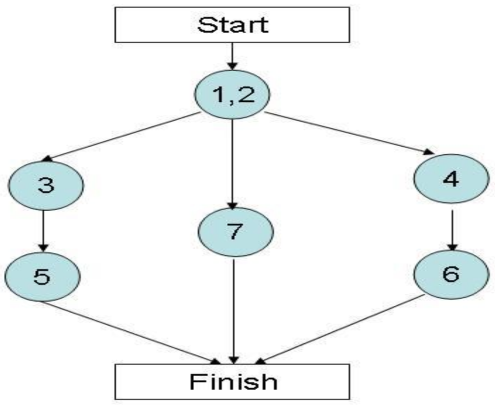


Fig. 3. The Data Dependence in a fixed iterative step

2.3. Inter-steps based data dependence in the Algorithm

We will take the data-dependence between step *k* and step *k+1* as example to show the inter-steps parallelism. 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*. In the operation '7', to get the value $B_{i,k}$ (*i=1…q and i≠q; k=1…q*) of *k+1* we must know the value $A_{i,k}$ (*i=1…q and i≠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 block "*X*" to block "*Y*" (*X, Y* is the block of matrix) signifies the operation on block "*Y*" should not be executed before the operations on block “*X*” are finished. Then we can describe the data dependence using directed arc (See Fig.5). ‘Loop 1’ in the Fig.5 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 4 shows us all the inter-steps data dependence existing between step 2 and step 3 with q is 5.

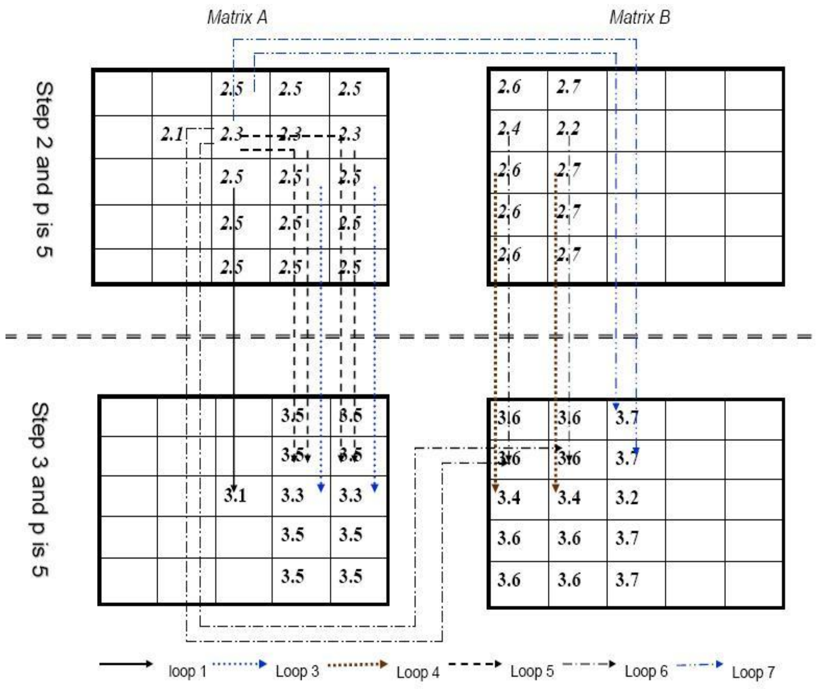


Fig.4. Inter-steps Data Dependence

We have analyzed all the inter-steps data dependence using Fig.4 and the intra-step based data dependence using Fig.3. Then we can summarize all the data dependence using Fig .5.

3. Formal Description of Data Dependence

The data dependence is up to write operation on blocks. It is the data operation that determine when and which blocks can be executed.

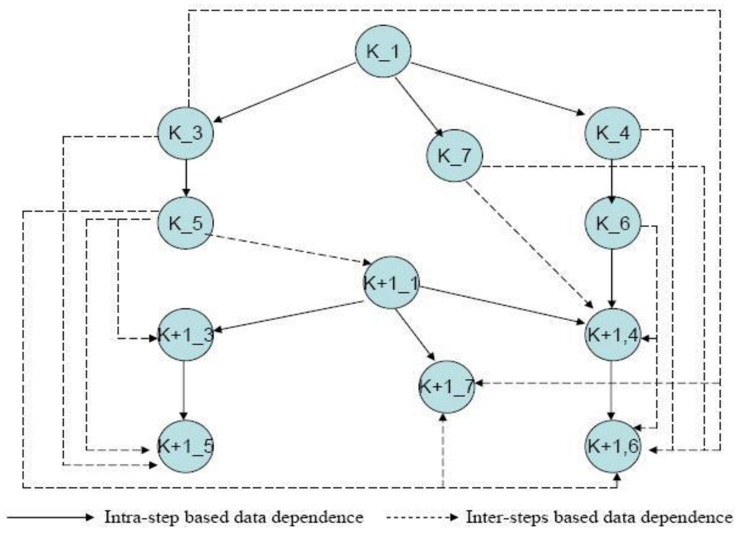
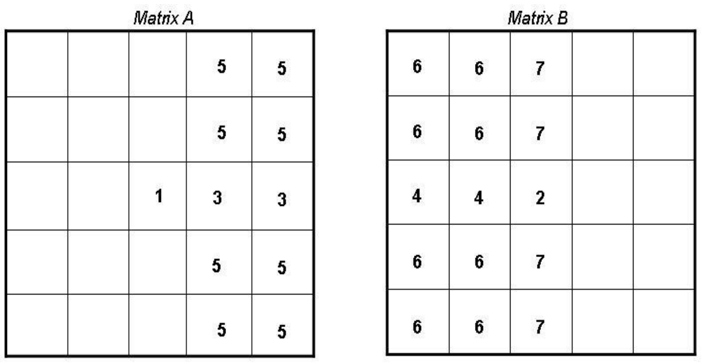

K_1
K_3
K_7
K_4
K_5
K_6
K+1_1
K+1_3
K+1,4
K+1_7
K+1_5
K+1,6
Intra-step based data dependence
Inter-steps based data dependence

Fig.5. All the Data Dependence in the algorithm. 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.


Matrix A

			5	5
			5	5
		1	3	3
			5	5
			5	5

Matrix B

6	6	7		
6	6	7		
4	4	2		
6	6	7		
6	6	7		

Fig.6. The data operation at step 2 with q is 5 in Algorithm 1

Fig.6 shows us that write operation on each block after executing one iterative step. These operations can be divided into two parts according to its goal written into (matrix *A* or matrix *B*). Operations ‘1’, ‘3’ and ‘5’ are on matrix *A*, while operation ‘2’, ‘4’, ‘6’ and ‘7’ are on matrix *B*.

It is the many similarities existing in write and read operation on matrix A and matrix B both in the intra-step parallelism and in the inter-steps parallelism that we will 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. Let us represent domain $AB^{k}_{i,j}$ using a triplet (k,i,j), then we can use the triplet KIJ = {(k,i,j) | k,i=1…q and j=k+1…k+q} to mark the status of block operation, i.e. KIJ represents the operation made at the step k in the row i and column j of augmented matrix $AB_{i,j}$. For the algorithm, one $AB^{k}_{i,j}$ is modified once and only once per iterative 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 ≯ 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 Fig.5 . The data dependence can thus be represented by the triplets defined above. For all the data dependence existing in Fig.5, we have:

Intra-step 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-step 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 relation obtained above according to operation marked in algorithm 1.
(k, k, k)≯(k, k, j)   k=1… p;   j=k+1…p   **(1)**
(k, k, k)≯(k, k, j) and (k-1, k, j)≯(k, k, j)   k=2…p;   j=k+1… p   **(3)**
(k, k, k)≯(k, k, j+q) and (k-1,k, j+q)≯(k, k, j+q) k=2… p; j=1… k-1 **(4)**
(k,k,j)≯(k,i,j) and (k-1,i,k)≯(k,i,j) and (k-1,i,j)≯(k,i,j) *k=2…q;j=k+1…q; i=1… q and i≠k*   **(5)**
(k,k,j+q)≯(k,i,j+q) and (k-1,i,j+q)≯(k,i,j+q) and (k-1, i, k)≯(k, i, j+q) k=2…q; j=1…k-1; i=1…q and i≠k   **(6)**
(k,k,k)≯(k,i,k+q) and (k-1,i,k)≯(k,i,k+q) k=2…p; i=1…p and i≠k   **(7)**

4. Design of Hybrid Parallel Programming Paradigm

As suggested by Ian Foster [13], parallel processing on an algorithm needs 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.

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, referring to Fig.1 and Fig.2. After partitioning, the objects of all the operations in the algorithm focus on blocks (sub-matrices), which makes a more complex problem 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 blocks 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 data-dependences on different blocks determine the sequence of different operations on a block (sub-matrix). Fig.1 shows us the inter-step data dependence in Matrix *A* and 12 blocks has the same write-operation (operation '5'). They have the same operation and there is no communication between them. I.e. 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.4, the operation represented by loop '1' shows us that before the 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.

As to the stages of agglomeration, mapping. Because all the write-operations of the algorithm are based on sub-matrix and the results will be return to the Matrix *A* after any a step of operation, so agglomeration is finished when all the operations on sub-matrix are end. After the hybrid programming paradigm, as much as sub-tasks will be generated and you can use appropriate schedule stratagem to map the sub-tasks to computing resources.

Next, a Flow-chart of hybrid programming paradigm can be seen in Fig.7. Based on the analysis above, a formal description of parallel programming paradigm of BbGJ for desktop grid platform can be presented using Algorithm 2 in this paper.

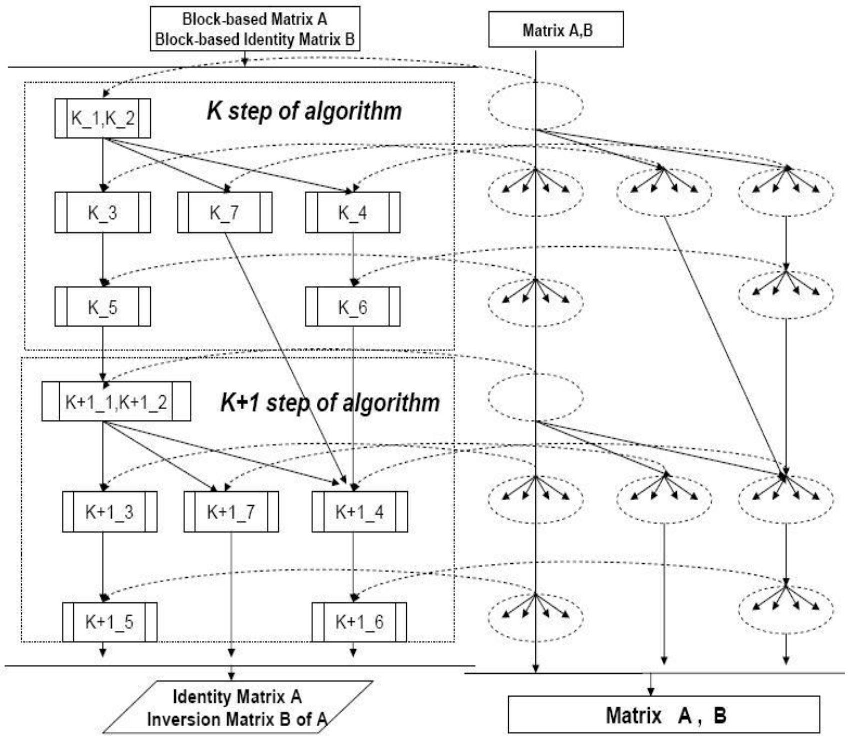


Fig.7. Left part is the flowcharts of the paradigm execution and the right part is the corresponding data operation. q steps needed to execute in the paradigm and the execution between step 'k' and step 'k+1' is described. In the right parts, 'many arrows' represents these data can be executed simultaneously.

The series of three tuples in new hybrid program paradigm for desktop grid system can be described using Fig.8. The Fig.9 shows us the workflow of different operations in the algorithm. The execution can be described using a series of three tuples. The key path of those three tuples is the base of setting the propriety of different operation. Three-level based propriety is set in the hybrid algorithm for desktop grid system.

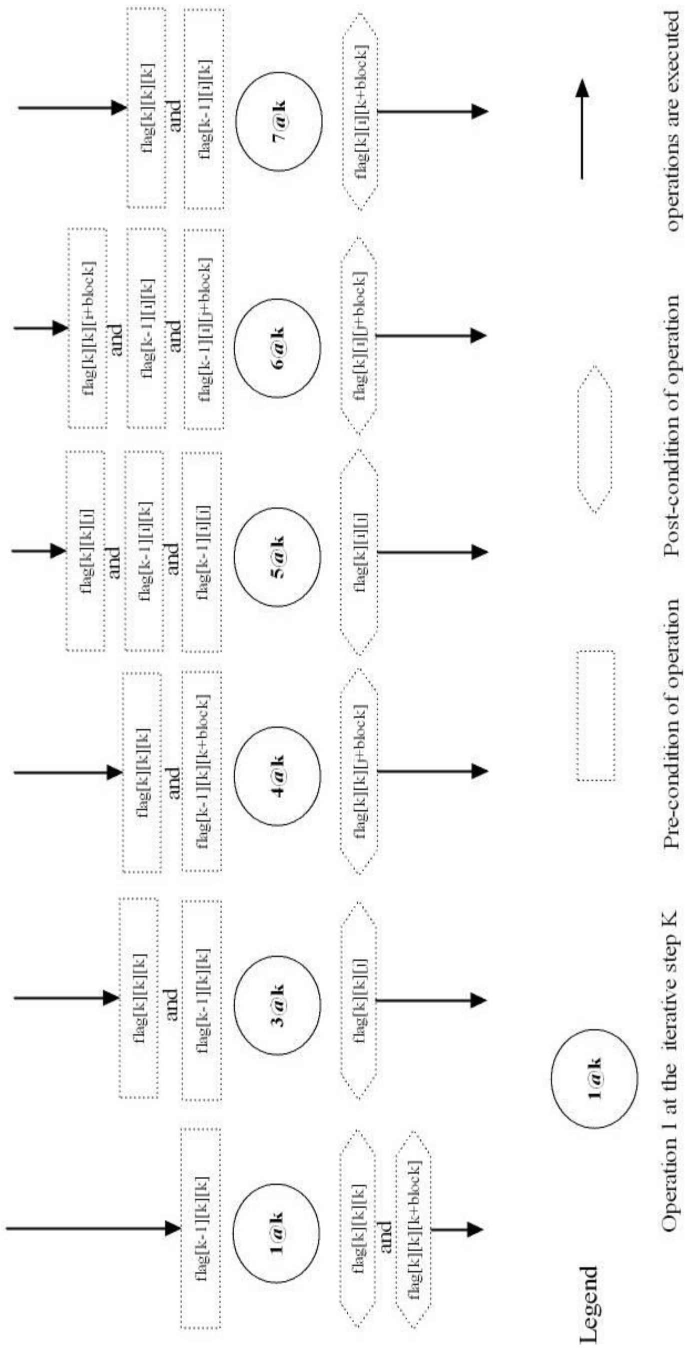


Fig.8. The series of three tuples of BbGJ in desktop grid system

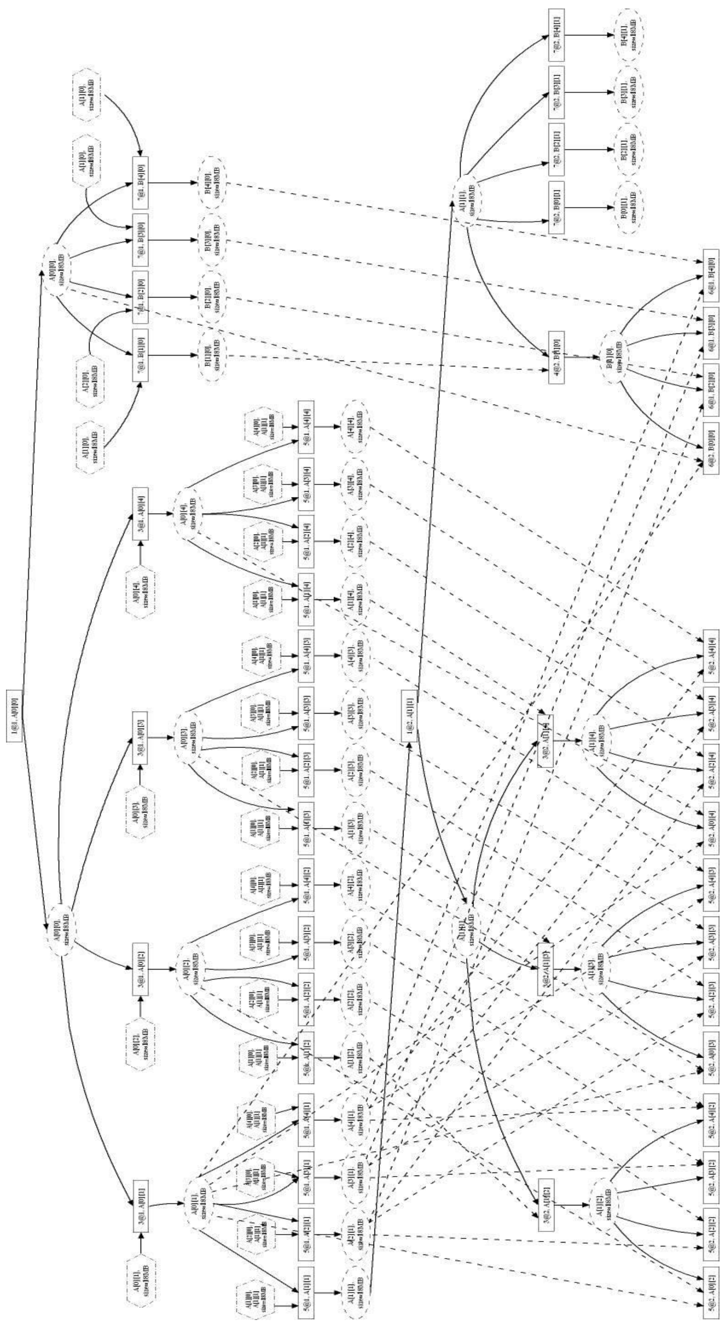


Fig.9. Workflow of three tupes in the BbGJ

```
Hybrid parallel algorithm for desktop grid system
------------------------------------------------------------------------
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    $A^{k}_{k,k} \leftarrow (A^{k-1}_{k,k})^{-1}$;
the priority of this operation is set 1(highest level)
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 priority of this operation is set 1(highest level)
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 priority of this operation is set 2 (important level)
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 priority of this operation is set 1 (highest level)
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 priority of this operation is set 2 (important level)
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,...,q; i=1,...,q and $i \neq k$)
the priority of this operation is set 3 (normal level)
the logical value of (k, i, k+q) is set to be true.
end if
------------------------------------------------------------------------
```

5. Performance Evaluation

In this section, we first evaluate the performance of traditional BbGJ algorithm in real desktop grid environment. The second experiment is to show the performance of our adapted algorithm in real desktop grid environment. Then, we compare the performance of our algorithm and traditional one in ideal desktop environment. All this experiments are based on YML/XtremWeb [5][6][9] (middleware) on desktop grid platform through the comparison the hybrid paradigm with intra-step parallel programming paradigm[3][7][15]. To state conveniently, we call the hybrid parallel algorithm proposed in this paper for desktop grid environments as “BbGJ_DG”. Here what we want to emphasize is that all the experiments are just to show the performance of hybrid programming paradigm, no special middleware is necessary, you can choose any middleware what you like. In our experiment, the topology of computing resources can be described using Fig.10 and the computational resources can be described as follows:

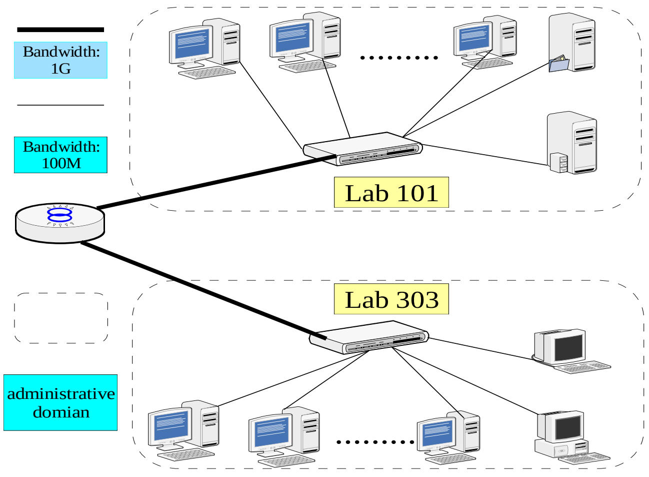


Fig.10. The topology of desktop grid platform

Table 1. Resources in Desktop Grid platform

Site	Node	CPU/Memory
Lab 303	16	Inter 2.66GHz/512M
Lab 101	64	AMD 1.8 GHz/512M

5.1. The character of desktop grid environment

Experiment motivation: test the performance of traditional algorithm in the desktop grid environment.

Experiment Environment: the first case is 30 nodes based desktop grid platform in Lab 101. The experiment time is set in the morning when the computing resources are high volatile. The second case is based on 30 nodes from Lab 101 (20 nodes) and Lab 303 (10 nodes)

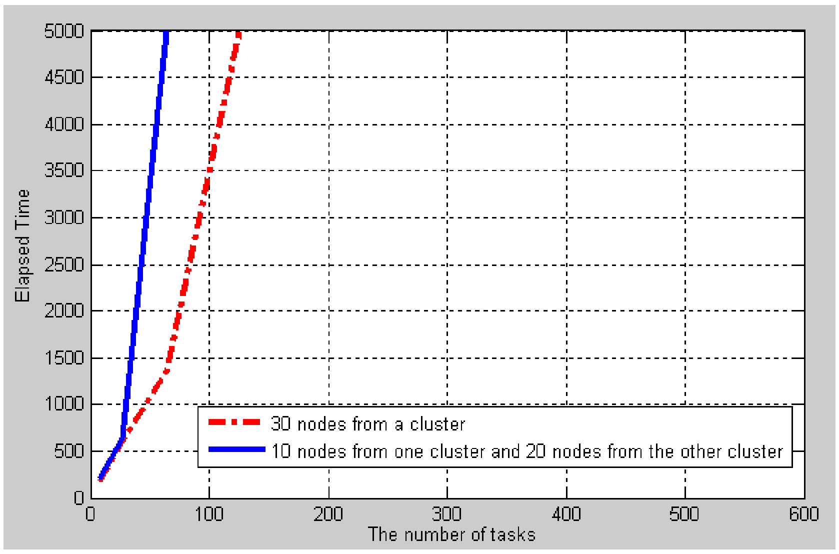


Fig.11. The performance of traditional version in desktop grid environment

As we mentioned above, the key character of desktop grid environment is its volatility [11,17] which causes frequent task migration between computing nodes. Paper [16] also emulate BbGJ algorithm in the desktop grid environment and the conclusion is that perhaps 163 days can finish the whole program. This experiment is also to evaluate the performance of data dependence based algorithm in desktop grid environment. In this experiment, we suppose when the execution time is more than 5000 (s), we think the program can't be executed successfully, i.e. the program can be finished in this environment.

Fig.11 shows us the when the number of tasks is not large, the program can be finish and when the number becomes larger, the program can't be executed successfully. From this experiment, we can conclude that when there is data dependence between operations in the parallel algorithm, we can't get better performance using directly adapted version of parallel algorithm. It is better for us to take volatility into consideration and multi-copies of the task have to be sent to more computing nodes. Also, the more important tasks which is in the key path have the priority to be executed. Only those factors are considered during allocating tasks to computing nodes, the better performance can be achieved. The next experiments just to show the new hybrid parallel algorithm proposed in the paper which takes those factors presented above can get better performance.

5.2. The performance of "BbGJ_DG" in desktop grid environment.

Experiment motivation: test the performance of "BbGJ_DG" in desktop grid environment.

Experiment Environment: the first case is 30 nodes based desktop grid platform in Lab 101. The experiment time is set in the morning when the computing resources are high volatile. The second case is based on 30 nodes from Lab 101 (20 nodes) and Lab 303 (10 nodes)

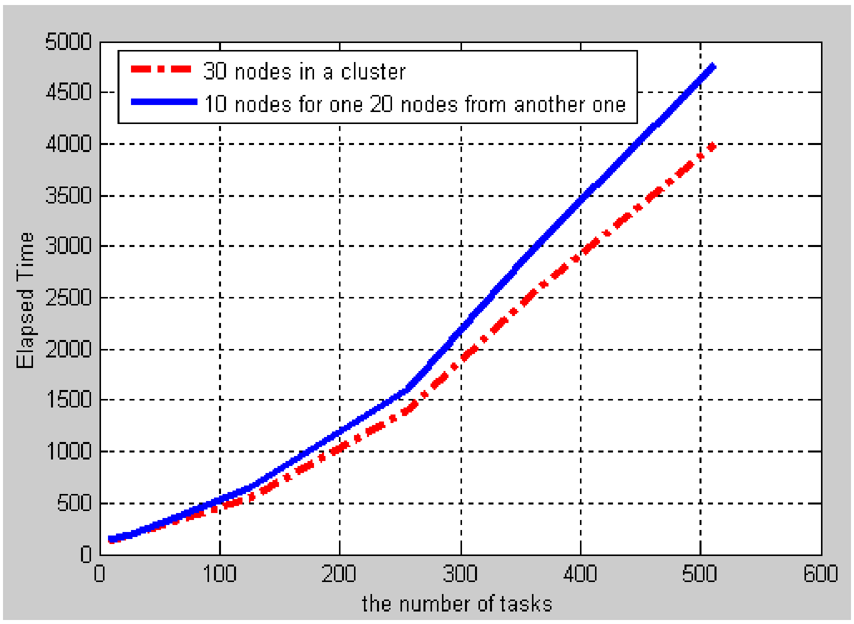


Fig.12. The performance of "BbGJ_DG" in desktop grid environment

The experiment in section 5.1 show that traditional version of BbGJ can't adapt to desktop grid environment and in more case, the program can't be executed successfully. In this paper, a new parallel adapted version which is called as "BbGJ_DG" takes the character of desktop grid environment into consideration. The experiments testify that, BbGJ_DG can be executed successfully on time. The program can be finished in two cases. So the conclusion can be achieved that BbGJ_DG can adapt to desktop grid environment. The difference between two cases in this experiment is from the experiment environments, which is cluster based desktop grid environment and while the other is grid based desktop grid environment.

5.3. Time difference between BbGJ_DG and traditional one

Experiment motivation: test the performance of BbGJ_DG 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: 45 nodes in Lab 101 and 15 nodes in lab 303. The experiment time is set at midnight (begin at 0 o'clock) at which all the computing resources can be seen as dedicated computing resources. This environment can be seed as ideal environment.

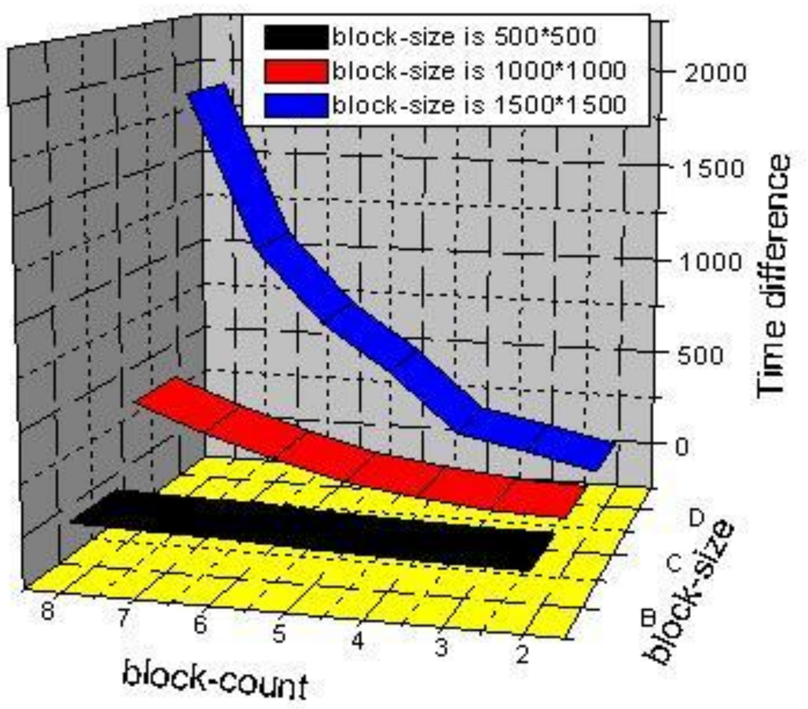


Fig.13. Time difference becomes very large when the block-size of matrix increase

From the Fig.13, we can find that when the block-size is not large, the elapsed time of algorithm based on BbGJ is just a little less than that based on intra-step paradigm (traditional parallel algorithm). And time gap becomes larger with the increase of block-size. The reason is that when block-size is small, more time is consumed in communication and schedule while time used to compute is small. So the advantage of BbGJ_DG is not obvious. From the analysis in section 3 and 4, we know the advantage of BbGJ_DG is to make full use of computing resources in desktop grid platform, i.e. the BbGJ_DG can make all the tasks available to be executed to be executed in parallel. With the increase of block-size, the advantage of BbGJ_DG becomes more and more obvious. the similar conclusion can be achieved when changing block-count and fixing the block-size

Time difference becomes larger with the condition of block-count change. With the increase of block-size, the computing time takes a great proportion and better concurrency in the BbGJ-DG ensures maximizing parallelism of program execution, which shortens 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 BbGJ_DG, 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.

6. Conclusion

This paper presents a hybrid parallel adapted version of BbGJ for desktop grid environment based on the analysis of data dependence in the algorithm. The new 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 desktop Grid platform in which more computing resources can be harnessed. At the same time, BbGJ_DG take the character of desktop grid system into consideration. So BbGJ_DG can get better performance in desktop grid environment. The experiment also testify the good performance when solving a real problem, and its scalability makes sure the paradigm can be tailored to more computing resources in desktop grid environment very easily.

Acknowledgements. This paper is supported by the Open Foundation of State Key Laboratory of Hydrology-Water Resources and Hydraulic Engineering (Grant No. 2011490804).

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):167–185, (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 350–354, (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 362– 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 25–32, (2006).
7 L. M. Aouad and S. G. Petiton. Parallel basic matrix algebra on the grid'5000 large scale distributed platform. In CLUSTER, (2006).
8 F. Cappello et al: Grid'5000: a large scale and highly reconfigurable grid experimental testbed. In The 6th IEEE/ACM International Conference on Grid Computing, pages 99–106, (2005).
9 F. Cappello, S. Djilali, G. Fedak, T. Herault, F. Magniette,V. N′ eri, and O. Lodygensky. Computing on LargeScale Distributed Systems: Xtremweb Architecture, Programming Models, Security, Tests and Convergence with Grid. Future Generation Computer Science, (2004).
10 S. T. Ian Foster, C. Kesselman. The Grid: Blueprint for a Future Computing Infrastructure. Morgan Kaufmann, USA, (1999).
11 Ling Shang, Zhijian Wang, Xiaofeng Zhou, Xiaoping Huang, Yongshang Cheng: TM-DG: a trust model based on computer users' daily behavior for desktop grid platform . ACM and ACM SIGPLAN Proceedings of the 2007 symposium on Component and framework technology in high-performance and scientific computing, Montreal, Quebec, Canada (2007)
12 Miquel Pericàs, 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, Jun 24-27, 2008, Toulouse, France (2008).
16 S. Petiton and L. Aouad. Large scale peer to peer performance evaluations, with Gauss-Jordan method as an example. Proceedings of the PPAM2003, fifth international conference on parallel processing and applied mathematics. Poland, Sep. 2003. Lecture Notes in Computer Science 3019, 938-945, (2004).
17 Derrick Kondo, Gilles Fedak, Franck Cappello, Andrew A. Chien, and Henri Casanova. 2006. On Resource Volatility in Enterprise Desktop Grids. In Proceedings of the Second IEEE International Conference on e-Science and Grid Computing (E-SCIENCE '06). IEEE Computer Society, Washington, DC, USA, 78-(2006)
18 Derrick Kondo Eric Wing. 2002. Models and Scheduling Mechanisms for Global Computing Applications. In Proceedings of the 16th International Symposium on Parallel and Distributed Processing (IPDPS '02). IEEE Computer Society, Washington, DC, USA, (2002)
19 D.P. Anderson and G. Fedak. The Computational and Storage Potential of Volunteer Computing, IEEE/ACM International Symposium on Cluster Computing and the Grid, Singapore, May 16-19, (2006).

Yizi Shang is a research assistant of hydro-informatics at Tsinghua University. He received B.Sc. degree in computers and electrical engineering from North China University of Water Resources and Electric Power in 2005, and the M.S and Ph.D degree in hydro-informatics from Tsinghua University in 2010. He was a research fellow of school of engineering and applied sciences (SEAS) at Harvard University from 2008 to 2009. His research interests concentrate on the application of information and communications technologies in addressing the increasingly serious problems of the equitable and efficient use of water for many different purposes.

Guiming LU is a professor at school of information engineering in North China Universtiy of water conservancy and electric power. His research and interests cover distributed computing, parallel computing, Grid computing.

Ling SHANG is an associate professor at school of information engineering in North China Universtiy of water conservancy and electric power. His research and development interests cover distributed computing, parallel computing, Grid computing and Cloud computing.

Guangqian Wang is a professor of Department of Hydraulic Engineering at Tsinghua University. His areas of expertise include hydro-informatics and computation fluid. He has published more than 100 technical papers and authored or coauthored several books in his areas of expertise. Dr. Wang is also the director of the State Key Laboratory of Hydro-science and Engineering, Tsinghua University.

Received: September 07, 2010; Accepted: January 16, 2011.