International Journal of Grid Distribution Computing
Vol.7, No.6 (2014), pp.257-274
http://dx.doi.org/10.14257/ijgdc.2014.7.6.21

QTBiCGSTAB Algorithm for Large Linear Systemand Parallelized

Xiaomei Xu[1], Jilin Zhang*[2], Jian Wan[2], Li Zhou[2] and Ming Jiang[2]

[1]*Department of medical technology, Zhejiang ChineseMedical University, Hangzhou, China*
[2]*Department of Computer Science technology, Hanzhou Dianzi University,Hangzhou, China; Key Laboratory of Complex Systems Modeling and Simulation Ministry of Education, Zhejiang Provincial Engineering Center on Media Data Cloud Processing and Analysis*
xuxiaomei@zcmu.edu.cn, jilin.zhang@hdu.edu.cn

Abstract

According to the traditional stabilized biconjugate gradient algorithm (BiCGSTAB) deficiency in data locality, this paper proposed a QTBiCGSTAB algorithm whose core idea is that recursively divides sparse matrix with quarter tree into sub-matrix and reorders them, to improve the hit ratio of cache and enhance the algorithm's efficiency. And the idea is good for algorithm being parallized, that is proved by the numerical experiments later. It mainly shows, firstly, QTBiCGSTAB algorithm is more efficiency than BiCGSTAB, and the speedup would reach 1:330. The target division length would be influnced on the algorithm's performance; Secondly, for large linear system, parallized QTBiCGSTAB is more efficiency than serial's.

Keywords: *QTBiCGSTAB, Data locality, Parallelized*

1. Introduction

Many scientific application, such as numerical simulation of computational fluid dynamics(CFD) [1-4] and simultaneous localization in robotics [5-8], *etc.*, need to solve large and complicated linear equations, that is difficult to be solved with direct methods [9, 10] which need more memory to solve. Now there are some iterative methods proposed[11-14], which need less memory, solve large and complicated linear equations more efficiently. However, there are some deficiencies of them. In paper [15], Kestler solved linear PDE's with multi-tree, this method is only fit the symmetric systems. Paper [16, 17] constructed preconditioners to accelerate the convergence rates, that would takes more time and more memory to compute. And some implementations [18] divided the matrix into sub-blocks with CHOLMOD [19]. The arithmetic operations of the linear systems would be performed efficiently with block sub-matrix, but it is only fit to the sparse symmetric positive definite matrix.

In this paper, firstly, we study the BiCGSTAB [20] algorithm for which solving large and unsymmetric systems fast and smoothly, and find that has many SpMV which would take much time. According to the rule of the SpMV, to enhance hit ratio of the Cache andimprovethe data locality of the algorithm, we recursively divided sparse matrix with quarter tree into sub-matrix and reordered them. Secondly, we proposed the QTBiCGSTAB algorithmand analysed some factors that would be influnced on the algorithm's performance, such as the target division length and the properties of the sparse matrix; Thirdly, we

ISSN: 2005-4262 IJGDC

proposedthe parallelized QTBiCGSTAB algorithm, that shows serial QTBiCGSTAB is easy tobe parallized, and we analysed the relationship between algorithm's performance and some factors such as the number of processors, the properties of the sparse matrix and so on; Lastly, we took some numerical experiments and proved that QTBiCGSTAB would have better performace compared with the BiCGSTAB, and could be parallized well.

This paper is organized as follows: in the Section 2, we proposed the QTBiCGSTAB and analysed its performance with BiCGSTAB; then the parallized algorithm is given in Section 3. In the following section, the numerical experiments are given. At last, we summarize the paper.

2. Serial QTBiCGSTAB Algorithm

This section, we proposed QTBiCGSTAB algorithm which is improved the data locality of the BiCGSTAB algorithm with quarter tree, to enhence the cache hit ration. It's well-known that the matrix vector multiplication is the core of the BiCGSTAB algorithm. The performance of the SPMV would be influenced on its performance. In this Section, we propose the struct of the quarter tree tovrecursively divide sparse matrix with quarter tree into sub-matrix and reorders them. Then apply it for QTBiCGSTAB algorithm, and at last compared with BiCGSTAB algorithm to analyze its efficiency.

2.1.Serial QTBiCGSTAB Algorithm

Serial QTBiCGSTAB Algorithm is shown as Algorithm1, which is the solution for the linear equations such as equation(1). In the algorithm, it suppose the initial gusses value is x_0, the max error is Emax.

$$b = Ax \tag{1}$$

where b is the result vector, x is the calculation vector and A is the matrix of which the size is$n \times n$.

Some details of the algorithm are described as follows:

1) Definition of the TreeNode

The definition of the TreeNode in the algorithm 1 shown as Figure 1, where *rStart, rEnd, cStart, cEnd* are the position for the tree node's area in the matrix; data is the non-zeroes of the tree node; *tlNode, trNode, dlNode, drNode* are respectively top-left, top-right, down-left, down-right children nodes of the tree node.

```
struct TreeNode{
        int rStart, rEnd, cStart, cEnd;
        Vector data;
        TreeNode tlNode, trNode, dlNode, drNode;
}
```

Figure 1. The Struct of the Quarter Tree Node

Algorithm 1 Serial QTBiCGSTAB algorithm

1: TreeNode *treeLeaves*;
2: int *leavesCount=treeLeaves.getCountOfLeaves()*;
3: **for** j← 0, leavesCount **do**
4: $r_0(j) = \text{treeLeaves.get(j)}x_0\text{(j)}$
5: **end for**

6: $\rho_0 = \alpha_0 = \omega_0 = 1$
7:$v_0 = p_0 = 0$
8: **for** i ← 1, n **do**
9:$\rho_i = \hat{r_0} r_{i-1}$
10: $\beta = (\rho_i/\rho_{i-1})(\propto_i/\propto_{i-1})$
11:$p_i = r_{i-1} + \beta(p_{i-1} - \omega_{i-1} v_{i-1})$
12: **for** j ← 0, leavesCount **do**
13: $v_i(j) = treeLeaves.get(j) p_i(j)$
14: **end for**
15: $\propto_i = \rho_i/(\hat{r}_0 v_i)$
16: $s_i = r_{i-1} - \propto_i v_i$
17: **for** j ← 0, leavesCount **do**
18: $t_i(j) = treeLeaves.get(j) s_i(j)$
19: **end for**
20: $\omega_i = (t_i, s_i)/(s_i, s_i)$
21: $x_i = x_{i-1} + \propto_i p_i + \omega_i s_i$
22: **if** $\Delta x_i < E_{max}$ **then** $STOP$
23: **end if**
24: $r_i = s_i - \omega_i t_i$
25: **end for**

2) Get tree Leaves

Suppose the target division length is *d*, recursively dividing the matrix *A* into sub matrix A_{rc}, where *r, c=0, d, .. n/d.* To get better efficiency of the division, if the sub-matrix hadn't any none-zero data, it won't be divided continuely, otherwise it won't stop until the sub-matrix's length is shorter or equal to the target division length. The division method would save the time to get treeLeaves, such as Agorithm 2.

The dividing process is detailed shown as Figure 2. In the Step 1, if A_{00}*.data.size() > 0* and A_{00}*.rEnd - *A_{00}*.rStart>d*, then should divide into sub matrix that shows as Step2; else dividing the $A_{(n/2)0}$. In the Step 2, if A_{00}*.data.size() > 0* and A_{00}*.rEnd - *A_{00}*.rStart>d*, then should divide into sub matrix that shows as Step3; else dividing the $A_{(n/4)0}$. Finally, we would get the sub matrix such as Step *m*. During the division process, if A_{ij}*.data.size() > 0* and A_{ij}*.rEnd –*A_{ij}*.rStart≤d*, thenadd into tLeaves. Theadding order is with column, as red line shown as Figure 3.

Algorithm 2 Get and reorder sub-matrix with target division length *d*

1: TreeNode *root*;
2: Vector *tLeaves*;
3: **function** getTreeLeaves(TreeNode n)
4: **if** $n.rEnd - n.rStart \leq d$ **then**
5:**for***j* ← *0,tLeaves.size()* **do**
6: **if** $tLeaves.get(j).cStart == n.cStart$ **then**
7: $tLeaves.insert(j, n)$;
8: **break;**
9:**end if**
10: **end for**
11:**else**
12:**if**$n.tlNode.data.size(\ \ ) != 0$**then**
13:getTreeLeaves(*n.tlNode*);

```
14:end if
15: if n.dlNode.data.size( ) != 0 then
16:getTreeLeaves(n.dlNode);
17: end if
18:if n.trNode.data.size( ) != 0 then
19:getTreeLeaves(n.trNode);
20: end if
21:if n.drNode.data.size( ) != 0 then
22:        getTreeLeaves(n.drNode);
23:end if
24:end if
25: end function
```

3) sparse matrix vector multiplication

There are many SpMV (sparse matrix vector multiplication) caculations in the algorithm 1 such as 4, 13, 18steps. In step 4, it traverses *tLeaves*to get non-zero elements caculating with corresponding data of vector $x_0(j)$, and then accumulates for vector $r_0(j)$. Suppose A_{00}, $A_{d0},\ldots,A_{(n-2d)0}$, $A_{(n-d)0}$are none-zero data, the sub-vector of the$x_0(j)$of the step 4 could be more time in the Cache, and then more would reduce the overhead of the frequent replacement cache block, to enhance the performance of QTBiCGSTAB algorithm. The process of the step 13, 18 is the same as Step 4.

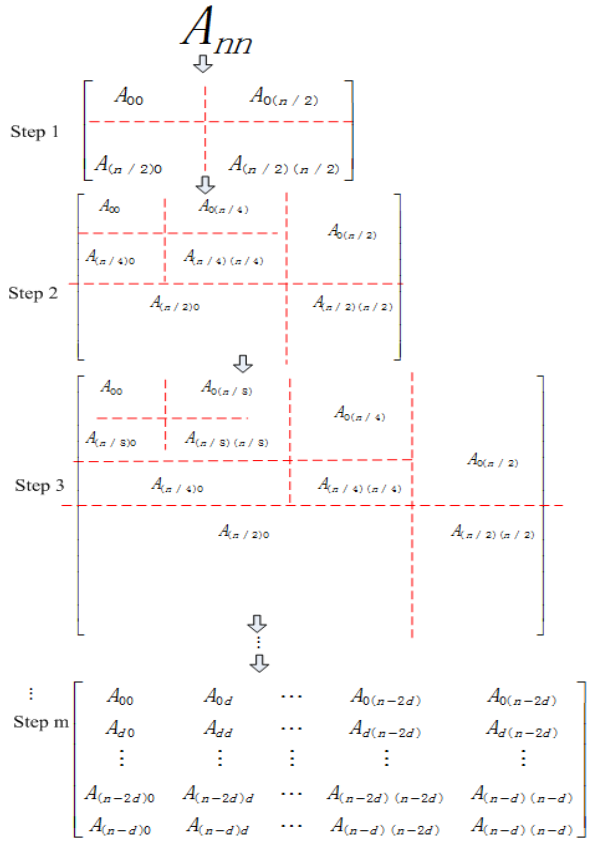

Figure 2. Recursively Dividing Matrix

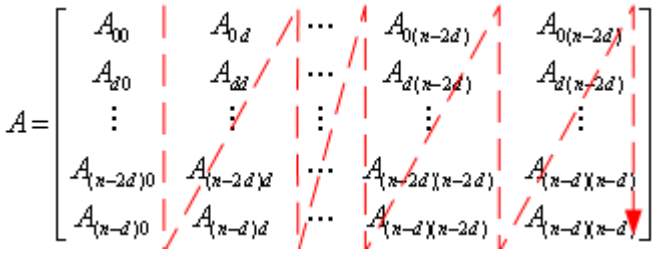

Figure 3. The Order of the Sub-Matrix Taken Part in the Operation

2.2. Analyze Efficiency of QTBiCGSTAB

This section,we would analyse the efficiency between BiCGSTAB and QTBiCGSTAB algorithm.

SpMV is the kernel of the BiCGSTAB, and the operation would take more time of the algorithm. Due to the sparse matrix is relatively large, the matrix difficultly read into the cache at once time. According to the locality principle of cache, data read order is greatly influenced on the performance of cache. So we would analyze reading amount between two algorithms.

Suppose *NNZ_COL(j), NNZ_ROW(i)*as nonzero elements amount of column *j*, row *i* in matrix *A*; $M_A$as the read amount of matrix *A*; M_x,*MQTx* as BiCGSTAB and QTBiCGSTAB algorithm's reading amount of vector *x*; M_b, MQT_b as BiCGSTAB and QTBiCGSTAB algorithm's reading amount of vector *b*; x_i is an element of the caculation vector *x*, $b_i$is an element of the result vector *b*; *M*, *MQT* are respectively as BiCGSTAB and QTBiCGSTAB's reading amount.

According to the rule of SpMV and the locality principle of cache, SpMV each time in BiCGSTAB algorithm, $x_i$need read *NNZ_COL(j)*times into cache, and each b_i read times M_{bi} shown as equation(2); So M_x;M_b;*M* are respectively shown as equation(3, 4, 5).

$$M_{bi} = \begin{cases} 1 & if\ NNZ_{row}(i) > 0 \\ 0 & if\ NNZ_{row}(i) = 0 \end{cases} \quad (2)$$

$$M_b = \sum_{i=0}^{n-1} M_{bi} \quad (3)$$

$$M_x = \sum_{i=0}^{n-1} NNZ_{COL(j)} \quad (4)$$

$$M = M_A + M_x + M_b \quad (5)$$

From equation (4, 5), we would get equation (6).

$$M = 2M_A + M_b \quad (6)$$

$$A_{rc} = \begin{bmatrix} a_{rc} & a_{r(c+1)} & \cdots & a_{r(c+d-1)} \\ a_{(r+1)c} & a_{(r+1)(c+1)} & \cdots & a_{(r+1)(c+d-1)} \\ \vdots & \vdots & \vdots & \vdots \\ a_{(r+d-1)c} & a_{(r+d-1)(c+1)} & \cdots & a_{(r+d-1)(c+d-1)} \end{bmatrix}$$

Figure 4. Elements of the Sub-Matrix Arc

Suppose the elements distribution of the sub-matrix A_{rc}such as Figure 4, where $r,c= 0, d, \ldots, n/d, d$ is the target division length, and a_{ij} is the elements of the sub-matrix A_{rc}, $i \in [r,\ r+d-1],\ j \in [c,\ c+d-1]$.

QTBiCGSTAB algorithm which reorders data of matrix A, would affect vector x_i;b_i reading times. According to the core idea of QTBiCGSTAB algorithm, the x_i of the caculatorvector x would finish all caculations, then be replaced out of the cache. So we could get each x_i read times MQT_{xi} shown as equation(7) and MQT_x shown as equation(8). Elements b_i read times is associated with the distribution of the none zero elements of matrix A and the target division length d. $M_{bi}(A_{rc})$ shown as equation (9) represents each sub matrix A_{rc} need read b_i times. $b_i$read times MQT_{bi} shown as equation(10), and MQT_b shown as equation(11). Finally, we would get MQT shown as equation (12).

$$MQT_{xi} = \begin{cases} 1 & if\ NNZ_{col}(j) > 0 \\ 0 & if\ NNZ_{col}(j) = 0 \end{cases} \tag{7}$$

$$MQT_x = \sum_{i=0}^{n} MQT_{Xi} \tag{8}$$

$$M_{bi}(A_{rc}) = \begin{cases} 1 & if\ \exists a_{ij} \neq 0,\ where\ a_{ij} \in A_{rc} \\ 0 & others \end{cases} \tag{9}$$

$$MQT_{bi} = \sum_{i=0}^{n/d} M_{bi}(A_{r(jd)}) \tag{10}$$

$$MQT_b = \sum_{i=0}^{n} MQT_{bi} \tag{11}$$

$$MQT = M_A + MQT_x + MQT_b \tag{12}$$

From above analysis, it must select a reasonable target division length d, so that the data would reside longer in the Cache to take part in operation. It means that would reduce the times of replace data block. In following section, we would analyze the efficiency of QTBiCGSTAB algorithm compared with BiCGSTAB from two factors both the target division length and the properties of matrix.

1) the target division length *d*

Suppose non-zeroes of the matrix *A* is well-distributed. From equations (9, 10, 11), the target division length *d* would be effect on MQT_b, and then would be influnced the efficiency of the QTBiCGSTAB algorithm. Suppose *d =1*, the $MQT_{xi}$and MQT_xwould be as equation(7, 8), MQT_bwould be as equation(13).

$$MQT_b = \sum_{i=0}^{n} NNZ\_ROW(i) \tag{13}$$

From equations (4, 2, 3, 7, 8, 13), we would get equation (14) and equation (15). And from equations (6, 12, 14, 15) we would get *MQT* equals to *M*, as equation (16).

$$MQT_b = M_x \tag{14}$$

$$MQT_x = M_b \tag{15}$$

$$MQT = M \tag{16}$$

From above analysis, if the target division length *d* is too short, it may lead to frequently replace the data of vector in the Cache. However, suppose target division length $d = n$, we would get such as $MQT_x = M_x$, $MQT_b = M_b$, and then get $MQT = M$. It means if *d* is too long, it may increase the ration of the cache capacity loss such as BiCGSTAB algorithm.

How to select reasonable target division length *d*. Suppose *SD*as the size of a nonzero element, *nz* as the number of the none-zero data in sub matrix A_{rc}, $nz = d^2$. To get better efficiency, in the ideal case, suppose the target division length d should meet the equation(17).

$$SD \times (d^2 + d + d) = cache\ capacity \tag{17}$$

2) non-zeroes distribution of the matrix.

From equation (7, 9), the non-zeroes distribution of the matrix would be effect on $MQT_x$and MQT_b, that would be effect on QTBiCGSTAB algorithm's performance. Suppose *d* is reasonable target division length, each $a_{ij} \neq 0$, where $a_{ij} \in A_{rc}$. The $MQT_b(A_{rc})$, $MQT_x(A_{rc})$, $M_b(A_{rc})$, and $M_x(A_{rc})$ are shown as equation(18, 19, 20, 21).

$$MQT_b(A_{rc}) = d \tag{18}$$

$$MQT_x(A_{rc}) = d \tag{19}$$

$$\mathrm{M}_b(A_{rc}) = d \tag{20}$$

$$\mathrm{M}_x(A_{rc}) = \mathrm{d}^{\,2} \tag{21}$$

Equations (18-21) shows QTBiCGSTAB replace time complexity is highly reduced, improve the cache hit rate, has better data locality, and more efficiency than BiCGSTAB.

However, suppose in sub matrix A_{rc}, at *r* row $a_{rj} \neq 0$, others $a_{ij} = 0$, we would get such as $MQT_x = M_x = d$,$MQT_b = M_b = 1$. In other words, the properties of matrix would affect on

the algorithm efficiency. In the experiment, we must consider the distribution of the sparse matrix's non-zero elements.

3) Time cost for dividing matrix

During thousands of iterations in the algorithm operation,the overhead of the dividing matrix would no longer be a key influence on the whole application performance compared to read data loss.

3. Parallelized QTBiCGSTAB Algorithm

To summarize the QTBiCGSTAB algorithm, we can find it mainly involves several calculation types as follow: (1) $y = Az$, where y, z are vectors, and A is a matrix; (2) $c = uv$, where u, v are vectors, and c is a numerical value; (3) $z = u + cv$, where u, v, z are vectors, and c is a numerical value; (4) $a = b \times c$, where a, b, c are numerical value. So to parallelize QTBiCGSTAB algorithm should parallelize above four kinds of calculations.

3.1.Algorithm Design

According to the principle of parallelization, we try to divide the calculation data on each processor which has relative independence data to calculate, that would reduce the communication among processors and decreases the Cache loss. The core idea of the QTBiCGSTAB is that recursively divide matrix into sub-matrix, which is consistent with distributedprocessing. So it can be rapidly deployed on processors, and can ensure the matrix block deployed with higher data locality. Assume the number of the processors is p, four types calculation of the QTBiCGSTAB algorithm are designed as follow.

1) Calculation Type (1)

Following the row direction, matrix would be divided into p blocks and each block has m_i $(where\ 0 \le i \le p-1)$ adjacent rows and then allocateto each processor in order. The matrix A has n rows, and n could be represented as equation (22). Define $index_i$as equation (23), where $i \in [0,p)$, $treeLeaves(i)$ is representedas the tree leaves on the ith processor, then the data on the ith processor is as below: $treeLeaves(i)$contains A_{rc}, where $r \in [index_{(i-1)}, index_i)$and $c \in [0, n)$, these nodes ordered with column which is same to the serial QTBiCGSTAB algorithm;caculate sub-vector $z[(index_{(i-1)}/d+1)\times d \ \ldots (index_i/d+1)\times d]$ and result sub-vector $y[(index_{(i-1)}/d+1)\times d \ \ldots (index_i/d+1)\times d]$, where d is the target division length and $index_{(-1)} = 0$.

$$n = \sum_{i=0}^{p-1} m_i \quad (22)$$

$$indexi = \sum_{j=0}^{i} m_j \quad (23)$$

The parallel algorithm of the Type (1) is shown as algorithm 3; PDATA contains the data of the ith processor. According to the execution steps of the QTBiCGSTAB algorithm, before caculating this type, it should wait the other processors to finish sub-vector z calculation, showing as step 2. At this step, it would overhead wait time. By the principle of data locality, parallelization can greatly improve the hit ratio of cache.

Algorithm 3 parallelized the Type (1) caculation

```
1: CPUDATA PDATA
2: wait(semz)
3: SpMV
```

2) Calculation Type (2)

Data on the *i*th processor is as follows: sub-vector $u[(index_{(i-1)}/d+1)\times d \ldots (index_i/d+1)\times d]$, sub-vector $v[(index_{(i-1)}/d+1)\times d \ldots (index_i/d+1)\times d]$and numerical value *c*.

The parallel algorithm of the Type (2) is shown as algorithm 4, where *PDATA* contains the data of the *i*th processor, such as sub-vector $u^T[(index_{(i-1)}/d+1)\times d \ldots (index_i/d+1)\times d]$ and $v[(index_{(i-1)}/d+1)\times d \ldots (index_i/d+1)\times d]$, the processor *id* and so on. According to the execution steps of the QTBiCGSTAB algorithm, each processor for this calculation type, needn't get data from the other processors. Each processor has relative independence to calculate this type. The value of *c* accumulate *temp*from each processorin step 4. If each processor has finished caculating as step 6, then post *semc* to other processors with step 10 to 12. The value *c* would be needed on the other processors to caculate next step of the QT-

Algorithm 4parallelized the Type (2) caculation

```
1: CPUDATA PDATA
2: temp= u^T[(index_(i-1)/d+1)×d ...(index_i/d+1)×d]v[(index_(i-1)/d+1)×d ...(index_i/d+1)×d]
3: lock(counterLock)
4: ptemp+= temp;
5: pcount--;
6: if pcount ==0 then
7: pcount ← p
8: c ← ptemp
9: ptemp ← 0
10: for i ← 1, p do
11: post(semc)
12: end for
13: end if
14: unlock(counterLock)
```

BiCGSTAB. So each processor need synchronously calculate to maintain data consistency.

The calculated data on each processor is relatively independent, can be very good for parallel computing. The algorithm parallelized would reduce the Cache loss and improve the data locality.

3) Calculation Type (3)

The data on the ith processor is as follows: sub-vector $u[(index_{(i-1)}/d+1)\times d \ldots (index_i/d+1)\times d]$, sub-vector $v[(index_{(i-1)}/d+1)\times d \ldots (index_i/d+1)\times d]$, vector $z[(index_{(i-1)}/d+1)\times d \ldots (index_i/d+1)\times d]$and numerical value *c*.

Algorithm 5 parallelized the Type (3) caculation

```
1: CPUDATA PDATA
2: wait(semc)
3:z[(index_(i-1)/d+1)×d ...(index_i/d+1)×d]=
```

u*[(index$_{(i-1)}$/d+1)×d ...(index$_i$/d+1)×d]+ c ×
v*[(index$_{(i-1)}$/d+1)×d ...(index$_i$/d+1)×d]*
4: **lock**(*counterLock*)
5: *ptemp*+= *temp*;
6: *pcount*--;
7: **if***pcount* == *0***then**
8: *pcount* ← p*;*
9: **for***i* ← *1*, *p***do**
10: post(*semz*)
11: **end for**
12: **endif**
13: **unlock**(*counterLock*)

The parallel algorithm of the Type (3) is shown as algorithm 5, where *PDATA* contains the data of the ith processor, such as sub-vector *z[(index$_{(i-1)}$/d+1)×d ...(index$_i$/d+1)×d]*and *u[(index$_{(i-1)}$/d+1)×d ...(index$_i$/d+1)×d]*, the processor *id* and so on. According to the execution steps of the QTBiCGSTAB algorithm, For the result vector *z[(index$_{(i-1)}$/d+1)×d ...(index$_i$/d+1)×d]*being needed for the SpMV on the other processors, it would post *semz* to other processors with step 9 to 11. During the process, each processor need synchronously calculate to maintain data consistency.

In this calculated type, calculated data on each processor is relatively independent, can be very good for parallel computing. The algorithm parallelized would reduce the Cache misses and improve the data locality. However, it would cost some waiting time for numerical*c*.

4) Calculation Type (4)

Data on the ith processor is as follows: numerical value *a*, *b*, *c*. In order to reduce the network communication overhead, each processor should calculatethis type.

3.2. Analysis of the Efficiency of the Parallelized Algorithm

In this section, we would analyze the parallelized QTBiCGSTAB algorithm. Suppose *MQT$_A$(i), MQT$_b$(i), MQT$_x$(i),* where *i ∈[0, p)* as the *i*th processor read amount of vector *A*, *b* and *x*. According to parallel algorithm, *MQT$_A$(i), MQT$_x$(i), MQT$_b$(i)* are shown as equation (25, 26, 27), where *rStart =(index$_{(i-1)}$/d+1)×d* and *rEnd = (index$_i$/d+1)×d.*

$$MQT_A(i) = \sum_{j=rStart}^{rEnd} NNZ_{ROW(j)} \tag{25}$$

$$MQT_x(i) = \sum_{j=rStart}^{rEnd} MQT_{xi} \tag{26}$$

$$MQT_b(i) = \sum_{j=rStart}^{rEnd} MQT_{bi} \tag{27}$$

And total read amout of the *i* processor *MQT(i)* as equation (28):

$$MQT\ (i)= MQT_A(i)+ MQT_x(i)+ MQT_b(i) \tag{28}$$

Each iteration of the parallelized QTBiCGSTAB algorithm's time overhead includes reading time, computing time and communication time. We assume each non-zero elements of matrix *A* computing time is $\tau_{compute}$, reading each data need time is τ_{read} and communication time is τ_{com}. Then the *i*th processor overhead total time $TPQT_i$ is shown as equation(29).

$$TPQT_i = MQT_A(i) \times \tau_{compute} + MQT(i) \times \tau_{read} + \tau_{com} \quad (29)$$

Then, we would get the total time TPQTof the parallelized QTBiCGSTAB algorithm as equation (30).

$$TPQT = \max TPQT_i, \text{ where } i \in [0, p) \quad (30)$$

According to the above assuming conditions and the serial QTBiCGSTAB's feature, we would get the total time TQTof the serial QTBiCGSTAB algorithm as equation (31).

$$TQT = MQT_A \times \tau_{compute} + MQT \times \tau_{read} \quad (31)$$

In following section, we would analyze the efficiency of parallelized QTBiCGSTAB algorithmcompared with QTBiCGSTAB from two factors both the number of processor *p* and the properties of matrix.

1) The number of processor *p*.

According to the parallel algorithm 2, each processor has m_i (where $0 \le i<p$) adjacent rows. if *p* is more larger, the $m_i$would be more smaller, and the calculation vector *z* on the ith processor should get more data from other processors, that result in increasing the value of the com and cause frequent Cache data replacement. We would detailed analyze as below.

$$MQT_A(i) = \frac{MQT_A}{p} \quad (32)$$

$$MQT_b(i) = \frac{MQT_b}{p} \quad (33)$$

$$MQT_x(i) = MQT_x \quad (34)$$

From equation (25, 27), the number of processor *p* would be effect on $MQT_A(i)$, $MQT_b(i)$. Suppose there isn't zero element in the matrix *A*, $MQT_A(i)$, $MQT_b(i)$, $MQT_x(i)$ are shown as equation (32, 33, 34).

From equation (28, 29, 30, 32, 33, 34), we would get equation as (35).

$$TPQT = \frac{MQT_A}{p} \times \tau_{compute} + \frac{M_r}{p} \times \tau_{read} + \tau_{com} \quad (35)$$

where $M_r = (p - 1) \times MQT_x + MQT$. To get better efficiency, in the ideal case, there would be $TPQT < TQT$, from equation (31, 35) then *p* should meet the formular (36).

$$\frac{1}{p} < 1 - \frac{\tau_{com}}{MQT_A \times \tau_{compute} + (MQT_A + MQT_b) \times \tau_{read}} \quad (36)$$

From above analysis, assuming that the size and sparse of the matrix unchanged, the

number of the processor would not be more is better. In practical applications, increasing the number of the processors, we need study the ratio change between caculation and the communication overhead. Therefore, determining the number of processors need study the specific task environment.

2) Properties of the matrix vector.

Each iteration of the QTBiCGSTAB algorithm, we assume the non-zeroes elements of the matrix A distributed densely, which are almost on the ith process. Equation (30) shows that the most overhead of the processor would play import role in the performance of the parallelized QTBiCGSTAB algorithm. So parallel QTBiCGSTAB algorithm must be considered that the number of non-zero elements on each processor, to effectly achieve well-balanced among processors and reduce communication overhead among them.

In practical application, the distributions of non-zero elements in sparse matrix are not balanced. So before the parallel algorithm, we try to make each processor load balancing calculated data to achieve the best overall performance value.

4. Numerical Experiments

The experiment environment is Intel(R) Xeon(R) CPU 5150 (Woodcrest Framework), frequency 2.66GHz, Bi-physical CPU, Bi-nuclear, 4GB Memory, runs CentOS 5.5 (Final) with 2.6.19 Kernel Version, and the GCC version number is 4.1.2.

We selected CFD matrices from Francois Pacull [21]for testing. Some information of these matrices would be showed as Table 1.

Table 1. The Basic Information of the Test Matrixes

name	rows	columns	non-zeroes	Non-zerons distribution
DK01R	903	903	11,766	
GT01R	7,980	7,980	430,909	
PR02R	161,070	161,070	8,185,136	
RM07R	381,689	381,689	37,464,962	

4.1. Efficiency of the Serial QTBiCGSTAB Algorithm

In this section, the matrixes shown in the Table 1 are tested to show the QTBiCGSTAB's performance associated with the target division length, and to prove that it is more efficient than BiCGSTAB algorithm.

1) target division length *d*

From Figure 5 to Figure 8. show the relationship between the efficiency of QTBiCGSTAB and the target division length *d*, where *X* direction represents the target division length *d*, *Y* direction represents time (ms) overhead with 3000 times iteration. From these figures, we can see target division length *d* is important influenced on the QTBiCGSTAB's performance,that is not too larger to better nor too small to better.

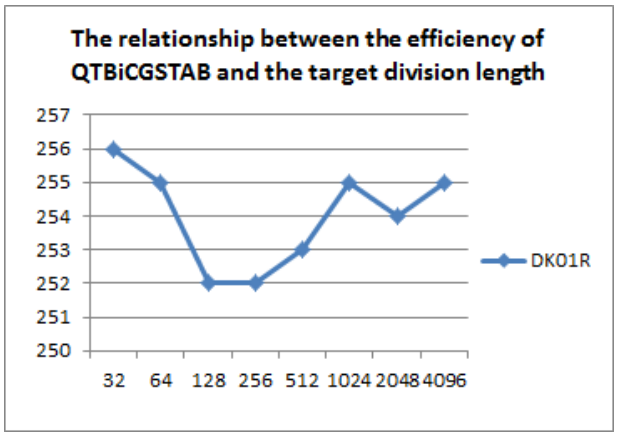

Figure 5. QTBiCGSTAB Algorithm with Different Target Division Length (1)

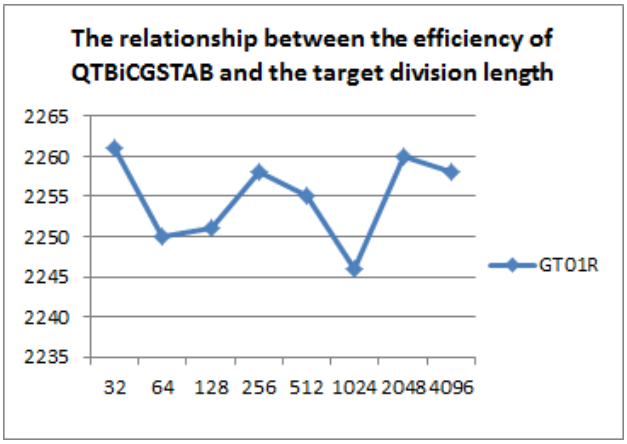

Figure 6. QTBiCGSTAB Algorithm with Different Target Division Length (2)

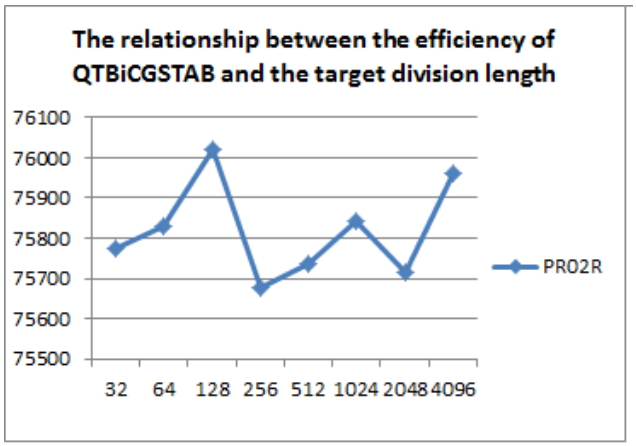

Figure 7. QTBiCGSTAB Algorithm with Different Target Division Length (3)

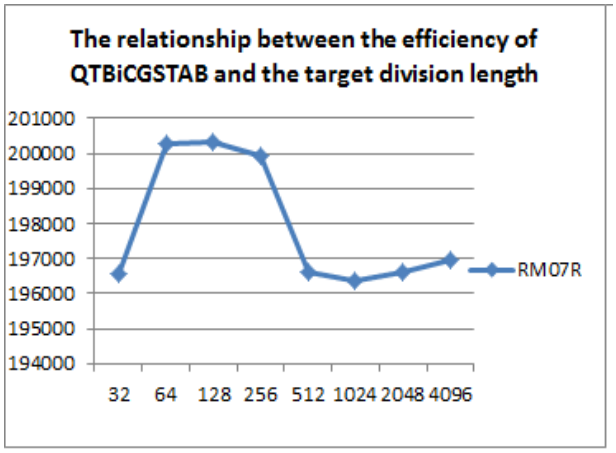

Figure 8. QTBiCGSTAB Algorithm with Different Target Division Length (4)

For the DK01R and PR02R matrix, their none zeroes distribute densely relatively, the reasonable division length is *256*. For the GT01R, RM07R matrix, the none zeroes are scatter relatively, lengthen target division length would get better efficiency, its reasonable division length is *1024*, with the length of the target division length *d* increasing, the nonzero elements in the Cache would stay longer time, It's reasonable target division length *d = 1024*. After reaching the reasonable division length, the target division length *d* is more longer, calculated vector replaced more frequently.

2) compared with BiCGSTAB algorithm

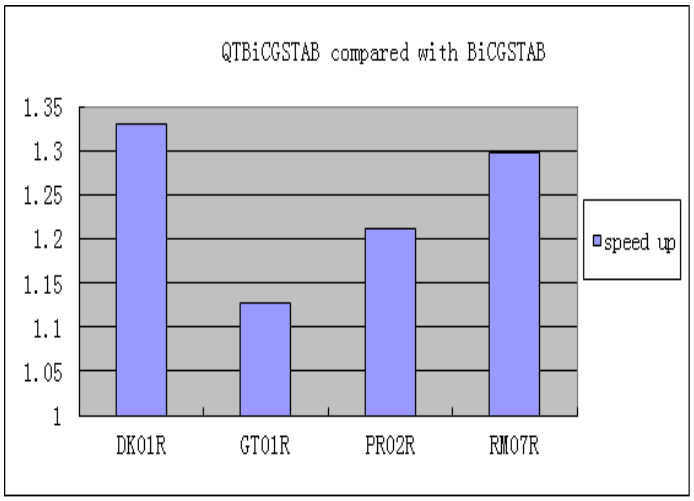

Figure 9. QTBiCGSTAB Compared with BiCGSTAB

From Figure 9, we can see that the QTBiCGSTAB has better performance than the BiCGSTAB. The result is that matrices shown in the Table I, selected reasonable target length. The highest speedup is 1.330 when processing the DK01R matrix with the two algorithms, and the lowest speedup is 1.127 when processing the GT01R matrix; the average speedup is about 1.241.

4.2.Efficiency of the Parallelized QTBiCGSTAB Algorithm

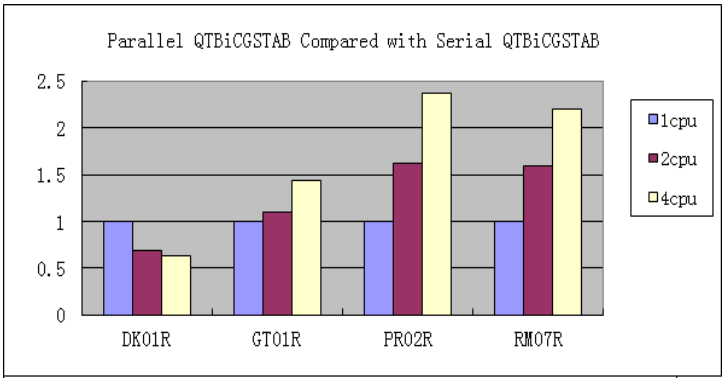

Figure 10. Parallelized QTBiCGSTAB Compared with Serial BiCGSTAB

The speedup of the parallel and serial algorithms with shared memory is shown in Figure 10. From Figure 10, we can see speedup of the matrixes such as GT01R, PR02R, RM07R, reaches a linear increase when the number of processors is increasing. For more non-zero elements of these matrixes having, the calculation overhead take mainly part, that result in good parallel performance. However, DK01R's speedup decreases with decreasing processor's amount. There are for some factors. First, DK01R matrix has less non-zeroes, only has 11766, and its dimension is smaller. Second, parallized algorithm involves the relevant data synchronization wait operation. During the process of the calculation, communication overhead take mainly part, it result in worst parallelized performance and the speedup of DK01R is less than 1.None-zeroes of GT01R matrix is more scattered, according to parallelized, none-zeroes on each processor would be not well-balanced. So PR02R matrix is more efficient than GT01R matrix.

5. Conclusions

In this paper, we proposed a serial QTBiCGSTAB algorithm and parallelized. Firstly, we presented a serial QTBiCGSTAB, then analyzed its performance with two factors both targetdivision length *d* and the property of the matrix. Secondly, a parallelized QTBiCGSTAB algorithm was proposed, and then analyzed its efficiency compared with serial QTBiCGSTAB. It found that processor's amount and the property of the matrix would be influnced on parallelized QTBiCGSTAB's performance. Finally, we selected some CFD matrices to do numerical experiments, to prove that QTBiCGSTAB is more efficient than BiCGSTAB, and the QTBiCGSTAB algorithm is easy to be parallized.

Acknowledgment

Supported by national science and technology support program(No.2012BAH24B04); National Natural Science Foundation(No.61202094) 2013-2015; Zhejiang Natural Science Foundation (No.Y13F020205); Zhejiang Provincial Engineering Center on Media Data Cloud Processing and Analysis(No.2012E10023-16); Higher School Visiting Scholars ProfessionalDevelopment Program from Zhejiang Department of Education(No.FX2013050); Zhejiang Provincial Technical Plan Project(No. 2011C13008).

References

[1] T. J. Chung, "Computational fluid dynamics", Cambridge university press, **(2010)**.
[2] S. F. C. Stewart, E. G. Paterson, G. W. Burgreen, *et al.*, "Malinauskas1Assessment of CFD Performance in Simulations of an Idealized Medical Device: Results of FDAs First Computational Interlaboratory Study", Cardiovascular Engineering and Technology, vol. 3, no. 2, **(2012)** June, pp. 139-160.
[3] A. G. Kuchumov V. Gilev and Popov, *et al.*, "Non-Newtonian flow of pathological bile in the biliary system: experimental investigation and CFD simulations", Korea-Australia Rheology Journal, vol. 26, no. 1, **(2014)** February, pp. 81-90.
[4] Q. Jin, X. Zhang, X. Li and J. Wang, "Dynamics analysis of bladder-urethra system based on CFD", Frontiers of Mechanical Engineering in China, vol. 5, no. 3, **(2010)** September, pp. 336-340.
[5] L. Polok, M. Solony, V. Ila, *et al.*, "Efficient implementation for block matrix operations for nonlinear least squares problems in robotic applications", Proceedings of the IEEE International Conference on Robotics and Automation, IEEE, **(2013)**.
[6] A. Martinelli, N. Tomatis and R. Siegwart, "Simultaneous localization and odometry self calibrationfor mobile robot", Autonomous Robots, vol. 22, no. 1, **(2007)** January, pp. 75-85.
[7] X. Zhang, A. B. Rad and Y. -K. Wong, "A Robust Regression Model for Simultaneous Localizationand Mapping in Autonomous Mobile Robot", Journal of Intelligent and Robotic Systems, vol. 53, no. 2, **(2008)** October, pp. 183-202.
[8] Zhiwei Liang, Songhao Zhu, Fang Fang, Xin Jin. Simultaneous Localization and Mapping in a Hybrid Robot and Camera Network System. Journal of Intelligent and Robotic Systems [J], 2010:1-24.
[9] I. I. Bosikova and N. A. Nedashkovskii, "Direct methods of solving systems of linear algebraic ef ions with complex a-matrices", Cybernetics and Systems Analysis, vol. 36, no. 2, **(2000)** April, pp. 276-286.
[10] G. Mayer, "Direct methods for linear systems with inexact input data", Japan Journal of Industrial and Applied Mathematics, vol. 26, no. 2-3, **(2009)** October, pp. 279-296.
[11] T. H. Chen and C. C. -P. Chen, "Efficient large-scale power grid analysis based on preconditioned Krylov subspace iterative methods", IEEE/ACM DAC in Proc, **(2001)** June, pp. 559-562.
[12] Z. Feng and P. Li, "Multigrid on GPU: Tackling power grid analysis on parallel SIMT platforms", IEEE/ACM ICCAD in Proc., **(2008)** November, pp. 647-654.
[13] Z. Feng and Z. Zeng, "Parallel multigrid preconditioning on graphics processing units (GPUs) for robust power grid analysis", IEEE/ACM DAC in Proc., **(2010)** June, pp. 661-666.
[14] J. Shi, Y. Cai, W. Hou, L. Ma, S. Tan, P. Ho and X. Wang, "GPU friendly fast Poisson solver for structured power grid network analysis", IEEE/ACM DAC in Proc., **(2009)** July, pp. 178-183.
[15] S. Kestler and R. Stevenson, "An efficient approximate residual evaluation in the adaptive tensor product wavelet method", Journal of Scientific Computing, **(2013)**, pp. 1-25.
[16] Z. Z. Bai, F. Chen and Z. Q. Wang, "Additive block diagonal preconditioning for block two-by-two linear systems of skew-Hamiltonian coefficient matrices", Numerical Algorithms, **(2013)**, pp. 1-21.
[17] C. E. Portugal, R. B. Prada and J. E. O. Pessanha, "ell-organized preconditioner for solving load flow problems by GMRES", 2011 International Conference and Utility Exhibition on Power and Energy Systems: Issues & Prospects for Asia (ICUE), **(2011)** September 28-30, pp. 1-6, doi: 10.1109/ICUEPES.2011.6497715.
[18] Z. Feng, X. Zhao and Z. Zeng, "Robust parallel preconditioned power grid simulation on GPU with adaptive runtime performance modeling and optimization", IEEE TCAD, vol. 30, no. 4, **(2011)**, pp. 562-573.
[19] T. Davis, "CHOLMOD: Sparse Supernodal Cholesky Factorization and Update/Downdate", **(2008)**, [online] Available: http://www.cise.ufl.edu/research/sparse/cholmod.
[20] H. A. Van der Vorst, "Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems", SIAM Journal on scientific and Statistical Computing, vol. 13, no. 2, **(1992)**, pp. 631-644.
[21] T. Davis, "The University of Florida sparse matrix collection", Nan Digest, vol. 97, no. 23, http://www.cise.ufl.edu/research/sparse/matrices, **(1997)** June.

Authors

Xiaomei Xu received the Master degree in Computer Software and Theory from Hanzhou dianzi University, Zhejiang, China, in 2006. She is currently a lecturer in Computer Science Technology in Zhejiang chinese medical University, China. She is interesting in research area both High Performance Computing and Cloud Computing.Now, she does some related researches in the Cloud Computing Research Center of Hanzhou Dianzi University.

Jilin Zhang received the Ph.D. degree in Computer Application Technology from BeijingScience Technology University, Beijing, China, in 2009. He is currently an associate professor in software engineering in Hangzhou Dianzi University, China. He is interesting in research area both High Performance Computing and Cloud Computing.

Jian Wan received his Ph.D. Degree in Computer Science from Zhejiang University, China, in 1996. He is now a professor and the dean of School of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou, China. His research interestincludes virtualization, grid computing, and services computing.

Li Zhou received her Master Degree from the School of Computer Science, Hangzhou Dianzi University, Hangzhou, China, in 2003. She is currently an associate professor in School of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou, China. She is with the Grid and Services Computing Lab and the Cloud Technology Research Center. Her current research interests include virtualization and grid computing.