<Final Report-18.337>

The parallelization of a block-tridiagonal matrix system for an electromagnetic wave simulation in TOKAMAK(TORIC) by MPI Fortran

Jungpyo Lee

Graduate Student in a department of

Nuclear Science & Engineering at MIT

Plasma Science & Fusion Center, Cambridge, MA, 02139

Office: 617-253-2374

jungpyo@mit.edu

05/13/2009

<Abstract>

The parallelization of Block-tridiagonal system in TORIC, an electromagnetic wave numerical simulation for TOKAMAK analysis, has been demanded for the reduced computation time. For the purpose, I implemented a new solver using a parallelization algorithm having both merits of the divide-and-conquer method and the odd-even cyclic reduction method. The new solver was tested with the current solver having a limited parallelization only in 2-dimension poloidal components. By adding 1-dimensional radial parallelization in the new solver, I obtained about two times faster computation speed and about fifty times smaller variance of the results. The use of full 3-dimension processor grid will be used to overcome the saturation of the improvement in the computation speed by parallelization in TORIC.

<Table of Contents>

1 Introduction

TORIC[1] is a MPI-Fortran based numerical code which has been developed to see the interaction between plasma and electromagnetic wave in TOKAMAK using a finite element method (FEM).(See Figure 1,2) The torus shaped geometry of TOKAMAK has three dimension variables along radial (ψ), poloidal (m) and toroidal direction. The solver for the code is expressed as

$$\underline{L_i} \cdot \vec{x}_{i-1} + \underline{D_i} \cdot \vec{x}_i + \underline{R_i} \cdot \vec{x}_{i+1} = \vec{y}_i \text{ for i=1,... } N_\psi$$

Each $\vec{x}_i$ is a $6N_m$ complex vector of poloidal Fourier components.
The master matrix is block-tridiagonal with $\underline{L_i}$,$\underline{D_i}$, $\underline{R_i}$ which size is $(6N_m) \times (6N_m)$.

In the current version of TORIC, it is implemented as a serial calculation for the tridiagonal system by Thomas algorithm along radial (ψ) direction with parallel block matrix operation for poloidal (m) direction on each radial component. When a square of the number of poloidal mode (N_m^2) is much larger than the number of processors, this implementation is very efficient. But, If the number of processor become comparable to N_m^2, numerous communication between processors deteriorate the performance of parallelization and reduce calculation speed. So, currently, the number of processors is limited as about $10^{-2} \times N_m^2$ by experience. In this project, I will modify TORIC to use more processors beyond the current limit and reduce the calculation time. One critical motivation to need a completion time as short as possible is to run TORIC several times in a big synthetic code for plasma analysis in a TOKAMAK, in the future. But, the current small limit for the number of processors may induce many free processors in a big cluster and retard the overall performance of the synthetic code.

When I use typical values of $N_\psi = 270$, $N_m = 255$ with 20 processors, the completion time to run TORIC one time is about an hour. My purpose is to reduce the time as an order of minutes by the use of about 1000 processors. In this sense, we need to make a parallelization of the radial direction as well as poloidal direction for the block-tridiagonal system.

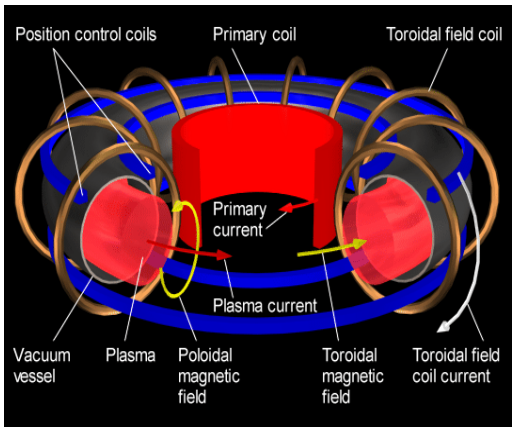


Figure 1. Schematic view of a Tokamak

[1] M.Brambilla, Plasma Phys.Control.Fusion 41(1999) I-34
w3.pppl.gov/theory/bin/PSACI-PAC03-Batchelor-Rev.ppt

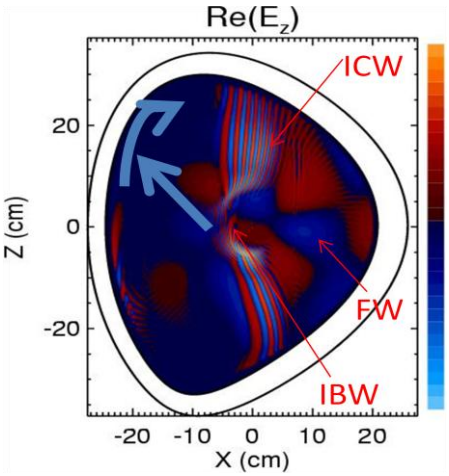


Figure 2 A result of TORIC(240N_r x 255 N_m) in a poloidal cross section of Tokamak. Two arrows indicates poloidal and radial directions. J. Wright, PSFC, PoP, 2004

2 Selection of an Algorithm

In this project, I needed to use a parallel algorithm for block-tridiagonal system which is scalable along radial direction instead of Thomas algorithm. Two representative algorithms for the system are a divide-and-conquer method and an odd-even cyclic reduction algorithm. When I evaluate the algorithms for TORIC, I have to consider both performance and compatibility with the current TORIC. The existing code use BLACS to parallelize the block matrix operation for each radial line. Unfortunately, in BLACS, there is no a block-tridiagonal built-in function but a simple tridiagonal factorization function, PDDTTRF, using the divide-and-conquer algorithm. There are more or less available free codes written in MPI-Fortran in internet, but they also need considerable modification for applying to TORIC.

Before implementing the parallelization routine, I compared many algorithms to choose one having a shortest computation time. Also, it should be applicable with the least modification of the existing code. In table 1, I summarize the calculation time and maximum memory required for each algorithm based on the reference 2, 3, and 4. The calculation times depend on many variables according to the algorithms.

	Cyclic odd-even reduction[2]	Divide and Conquer [3]	Thomas Algorithm ($P_1 = 1$)[4]
Calculation time for matrix operation	$(\log_2 P_1 + (\frac{n_1}{2P_1} - 1)) \times (13M^* + 6A^* + D^*)$	$\frac{n_1}{P_1}(6M + 4A + 2D) + P_1(3M + 3A + 2D)$	$n_1(3M + 3A + 2D)$
Maximum memory for a processor	$\frac{n_1}{P_1}(3n_2^2 + 2n_2)/P_2P_3 \times complex\ type$	$\frac{n_1}{P_1}(3n_2^2 + 2n_2)/P_2P_3 \times complex\ type$	$n_1(n_2^2 + 2n_2)/P_2P_3 \times complex\ type$

Table 1. Comparison of Block-Tridiagonal algorithms

$\boldsymbol{n_1 = N_\psi}: \#\ of\ radial\ components, \boldsymbol{n_2 = (6N_m)}: \#\ of\ poloidal\ components$

$Total\ \#\ of\ processors = P = P_1P_2P_3, \quad P_1 = dimension\ 1(radial)$

$M(\#\ of\ multiplcations), A(\#\ of\ additions) and\ D(\#\ of\ divisions) for\ a\ block\ matrix \sim O(\frac{n_2^2}{P_2P_3})$

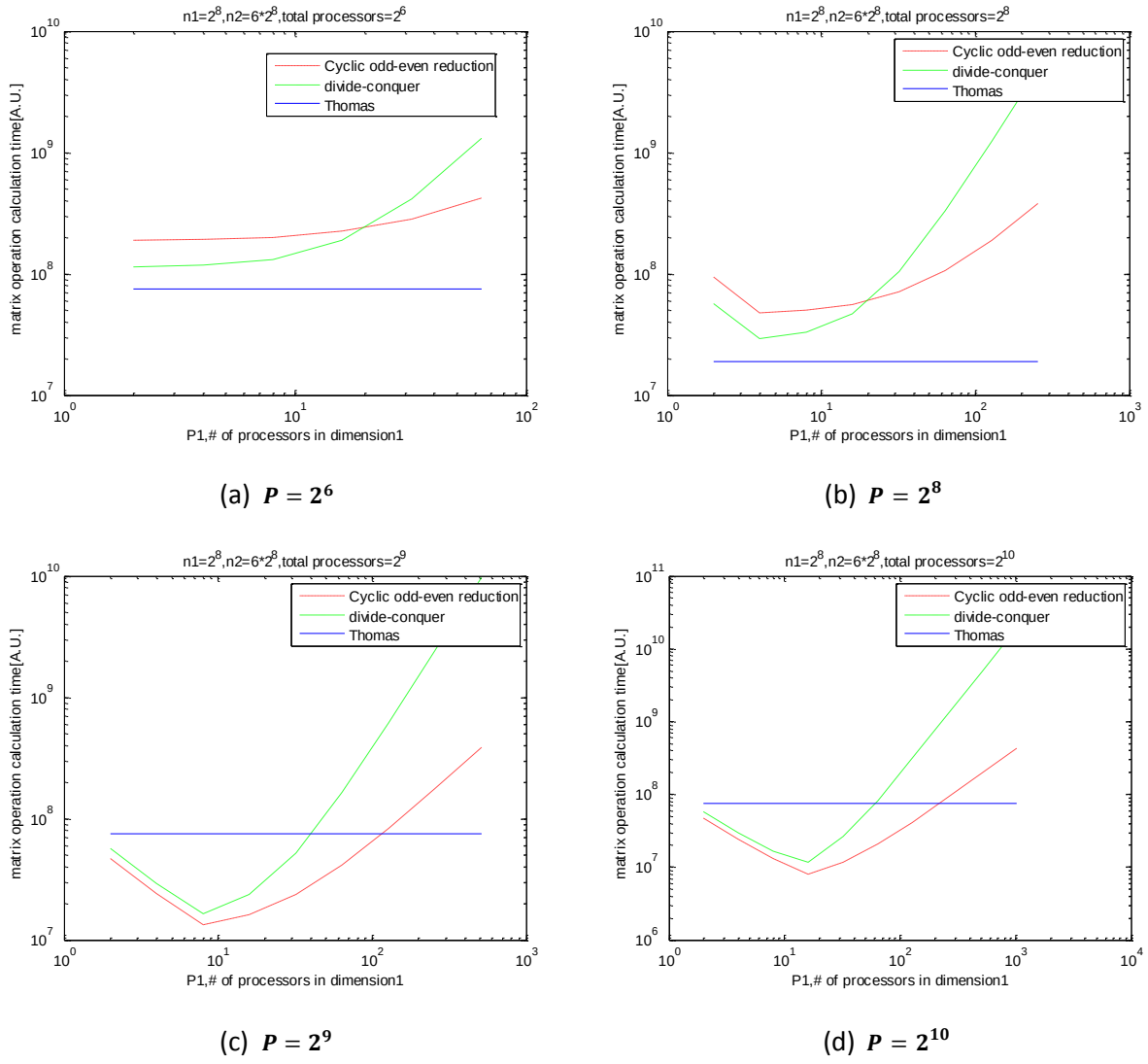
For the cyclic odd-even reduction algorithm, I assumed that both P_1 and n_1 are power of 2. In the reference 2, they assumed that the number of processors is big enough to be same as $\frac{n_1}{2}$, but it's not practical for our environment. So, I modified the factor of the calculation time in the reference 2 by including the serial calculation, $(\frac{n_1}{2P_1} - 1)$, before the reduction process which calculation time is order of $\log_2 P_1$. If I use a variable processor grid and transfer free processors of P_1 to P_2P_3 in the reduction

[2] H.S.Stone, ACM transactions on Mathematical Software,Vol1(1975),289-307
[3] H.H.Wang, ACM transactions on Mathematical Software,Vol7(1981),170-183
[4] http://en.wikipedia.org/wiki/Tridiagonal_matrix_algorithm

process, $M^*, A^*, and\ D^*$ can be reduced a lot, compared to $M, A\ and\ D$ in other algorithms. But, for convenience, all values of $M^*, A^*, and\ D^*$ as well as $M, A\ and\ D$ are assumed to be same.

For the divide and conquer algorithm, when comparing with the result in the reference 3, I don't count the operation which is independent with P_1 and n_1, because the values are small, an order of 1, whereas typical values of P_1 and n_1 are an order of 100. Also, I also selected not vector operation but matrix operation for blocks to compare the algorithms, because time for matrix operation is dominant in most algorithms.

In Thomas algorithm, due to the intrinsic characteristic of parallelization, as the number of processors (P_2P_3) become to be comparable to the number of components which can be parallelized, ($\mathrm{N_m^2}$), numerous communication between the processors deteriorate the calculation speed. So, I limited the maximum value of P_2P_3 as $(\mathrm{N_m}/32)\text{^}2$ by experience, when I calculated $M, A\ and\ D$



(a) $\boldsymbol{P = 2^6}$ (b) $\boldsymbol{P = 2^8}$

(c) $\boldsymbol{P = 2^9}$ (d) $\boldsymbol{P = 2^{10}}$

Figure 3(a)(b)(c)(d), the evaluation of calculation time for cyclic odd-even reduction, divide and conquer and Thomas algorithm.

As shown in figure 3, total number of processors is very important factor to determine the performance of algorithms. Below a certain threshold, such as $\boldsymbol{P = 2^8}$, using the Thomas algorithm with just 2-D poloidal parallelization for each matrix operation is the most efficient among the three algorithms because the additional parallelization along radial dimension indeed waste times for additional process such as making fill-in components when there are enough poloidal modes to be parallelized by the total number of processors. While the calculation speed of Thomas algorithm is not increased over the threshold, the cyclic odd-even reduction algorithm become faster than others as the number of processors are increased. The characteristic of reduction algorithm which speed is proportional to $\log_2 P_1$ become more beneficial. Also, we need to notice on the existence of optimized number for the dimension 1 (P_1) for both cylic reduction algorithm and divde-and-conquer algorithm. This result should be used in mapping 3-D grid in BLACS as shown in figure 4. By the result of the comparison, I decided to implement odd-even cyclic reduction algorithm for Block-tridiagonal solver in TORIC.

One available Fortran code for Block tridiagonal solver by Pascale Garaud, et al.[5] was a good reference for this project. Their code was originally implemented for the analysis on solar tacholine, and they used their own algorithm which was modified from the standard odd-even cyclic reduction algorithm to enhance the stability of the calculation[6]. I adopted their algorithm and wrote new subroutines compatible with the existing parallelization routine in TORIC.

[5] http://www.soe.ucsc.edu/research/report?ID=487
[6] P.Garaud, Mon.Not.R.Astron.Soc,391(2008)1239-1258

3 Code Implementations

3.1. Set up a 3-dimensional (3-D) processor grid

The common parallel block-tridiagonal solvers use 1-dimensional (1-D) processors grid, because a algorithm, whether it is divide-and-conquer or odd-even cyclic reduction, is easy to applicable to 1-dimensional processors grid, and the usual size of blocks, N_m, is much smaller than the number of blocks,N_ψ. However, in our case for TORIC, usually, N_m is big number as much as N_ψ, so that it needs to parallelize each matrix block as well as tridiagonal system, when considering the calculation speed. Therefore, in TORIC, it is good to keep the current 2-D processor gird for matrix operations with Blacs, and add 1-dimension for parallelize the tridiagonal system.

Another reason for 3-D processor grid is the concern for memory. If it depends only on 1-dimensional grid and the number of processors is limited by N_ψ, saving full blocks which size is $(6N_m) \times (6N_m)$ would be impossible for a processors. As the same reason, in current version of TORIC, the data in each block is distributed for 2-D grid processors.

The easiest way to implement 3-D grid is to use a context array in BLACS, which context has 2-D processors grid as it is already implemented. In BLACS, a context indicates a group within a boundary of a communication. In a context having full processors, it is possible to assign several sub-groups of processors corresponding to each context according to the specific maps (See an example in Figure 4)

```
*     Get default system context, and define grid
*
      CALL BLACS_GET(0, 0, CONTXT)
      CONTXT2=CONTXT
      CONTXT1=CONTXT
      CALL BLACS_GRIDINIT(CONTXT, 'Row', 1, NPROCS)

*     define grid map for contxt1 and 2
      IMAP1(1,1)=0
      IMAP1(1,2)=1
      IMAP1(2,1)=2
      IMAP1(2,2)=3
      IMAP2(1,1)=4
      IMAP2(1,2)=5
      IMAP2(2,1)=6
      IMAP2(2,2)=7
      CALL BLACS_GRIDMAP(CONTXT1, IMAP1,NPROW, NPROW, NPCOL)
      CALL BLACS_GRIDMAP(CONTXT2, IMAP2,NPROW, NPROW, NPCOL)
```

Figure4. The use of multi context for 3D processor grid in Blacs. In this example, total processors number is 8(2*2*2)

Using these contexts, I implemented 3-D grid which sub-groups are communicable each other when needed in tri-diagonal algorithm.

3.2. Divided forward reductions and Odd-even cyclic reductions

The algorithms by Pascale Garaud can be described as the combination divide-and-conquer method and odd-even cyclic reduction. Basically, the forward reductions are executed simultaneously in every group as a typical divide-and-conquer method. Except the first group, the reduction processes create fill-in matrixes at the last column of the previous group, as shown in figure 5(a). This fill-ins matrixes represent

"F _j,k" in step 1 of figure 5(b) explaining this reduction algorithm. Other operations making this upper diagonal matrix having "G_j,k" are same as that of Thomas algorithm.

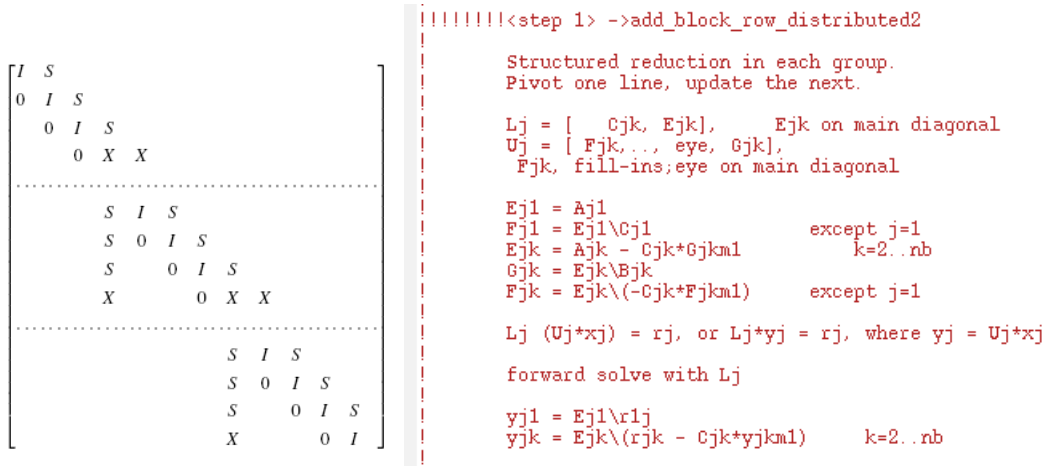


Figure5(a)(b). The full matrix illustration (a) and the code description (b) for the step 1 (The divided forward reduction).

After step 1, to reduce the number of communications between the groups for the eliminations fill-ins matrixes, I used odd-even cyclic algorithm. So, in step 2, I need to redistribute the fill-ins matrixes as the tri-diagonal forms composed of the first row in each group shown in figure 6(b). Before carrying out the redistribution, the matrixes in last row in each group should be transmitted to the next group. (See figure 6(a)). Then, the received right matrix which was "G_j-1,n" is eliminated by the appropriate linear operation with a following row and creating another matrix "T3'". This process in each group continues in sequence until the matrixes become the tri-diagonal forms in figure 6(b). This redistribution process is formulated in step 2 of figure 6. This step 2 is a process to prepare the cyclic reduction in step 3.

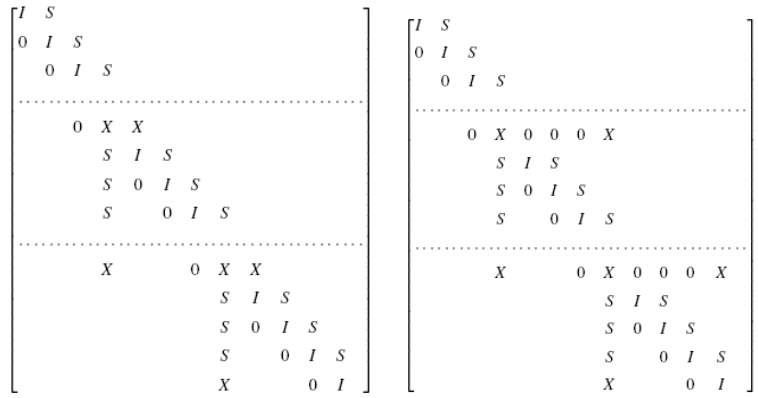


Figure6(a)(b). The full matrix illustration, before step 2 (a) and after step 2 (b).

The redistributed tri-diagonal forms can be reduced by a typical odd-even cyclic reduction in step 3 of the figure 7. This reduction was carried out in $\log_2 \frac{(P_1+1)}{2}$ steps (Compared to $\log_2 P_1$ in Section 2, this number is modified for this algorithm). This is, P_1 should be $2^n - 1$ instead of 2^n, and it requires the

total number of processor is several times of $2^n - 1$. This characteristic could be a weak point of this algorithm in practical computation environment, because a node in a cluster consists of 2^n processors typically. It may induce free processors in certain node always.

```
!!!!!!!!<step 2> ->pass_block_redistributed2
!
!       Structured reduction, second sweep step.
!       All groups pass their last blocks G,F and y
!       to the next,
!
!       initially,T1=F,T2=eye,T3=G,ycyc=yk
!
!       for k=2:nblocks-1
!       T2=T2-T3*Fk
!       ycyc=ycyc-T3*yk
!       T3'=-T3*Gk
!
!!!!!!!!<step 3> ->cyclic_reduction2
!
!       odd-even cyclic reduction for T1,T2,T3 and ycyc
!       :Invert, pass and update stage
!       [T1m T2m T3m         ] {xm}  {ym}    :odd
!       [    T1  T2  T3      ]*{x } ={y }    :even
!       [        T1p T2p T3p] {xp}  {yp}    :odd
!       when the number of groups=2**npgroup_exp-1
!       do the reductions in npgroup_exp-1 steps
!
!       (a)  invert matrices in all "odd" groups.
!            [T1m T2m T3m;ym]->[T2m\T1m eye T2m\T3m;T2m\ym]
!            [T1p T2p T3p;yp]->[T2p\T1p eye T2p\T3p;T2p\yp]
!
!       (b) send the blocks(T2m\T1m,T2m\T3m,T2m\ym) to the "next" group
!           (unless you're the "last" one)
!           send the blocks(T2p\T1p,T2p\T3p,T2p\yp) to the "previous" group
!           (unless you're the "first" one)
!       (c) All "even" groups receive and update blocks
!           T2'=T2-T1*T2m\T3m-T3*T2p\T1p
!           y'=y-T1*T2m\ym-T3*T2p\yp
!           T1'=-T1*T2m\T1m
!           T3'=-T3*T2p\T3p
!
!       After the last reduction, the form will be  T2'*x=y'
!
```

Figure7. The code description for the step 2 and step 3(Odd-even cyclic reduction).

3.3. Cyclic substitutions and Divided backward substitutions

In the end of the cyclic reduction in step 3, only one matrix T2 remains in a row, so I can obtain one component of solution by "x=T2\y". The value is substituted to find all other solutions in step 4 and step 5 as shown in figure 8. In step 4, the cyclic back substitution is executed in $\log_2 \frac{(P_1+1)}{2}$ step same as the cyclic reduction. Then, in each group, the serial back substitution continues simultaneously in step 5. In the step, each group except the first one should have the information of the solution in the previous group, "x_j-1", to evaluate the term contributed by fill-ins matrix "F_j,k" in the solution.

```
!!!!!!!!<step 4> ->cyclic_back_substituted2
!
!        Find the solutions x for the cyclic lines in all groups
!
!        x'=T2\(y-T1*xm-T3*xp) for every step(i=npgroup_exp-1 to 1)
!
!        So,for i=npgroup_exp-1 (i.e. one cyclic line)
!        x=T2'\y' (without xm and xp)
!
!        for i=npgroup_exp-2 (i.e. three cyclic line)
!        xm=T2m'\(ym'-T3m'*x)   (without xm)
!        xp=T2p'\(yp'-T1p'*x)    (without xp)
!
!        for the rest of the steps,
!        x'=T2\(y-T1*xm-T3*xp) (both xm and xp known)
!
!!!!!!!!<step5> -> ptri_solve2
!
!       Backward solve in each group
!       From step4, all yn in the groups are known
!       xjn = yjn              j=1..npgroup
!
!       x1k = y1k - G1k * x1kp1,    for j=1,k=n-1..1
!       xjk = yjk - Gjk * xjkp1-Fjk*xjmln,    for other j,k=n-1..1
!       note need to get Gk and Fk from disk in order of decreasing k
!       and receive xn from the previous group
```

Figure8. The code description for the step 4(cyclic substitution) and step 5(divided back substitution).

4 Result and Discussions

4.1. Test of the new solver in terms of processor grids

After implementing the algorithm, the new solver was tested with only use of dimension 1(P1) among 3D grid (i.e. $[\boldsymbol{P_1},\boldsymbol{P_2},\boldsymbol{P_3}]$ =[7,1,1] or [15 1,1]), in normal running condition of TORIC, $\mathrm{N}_{\psi} = 270$, $\mathrm{N}_{\mathrm{m}} = 255$. The results of electric fields and the derivatives of the fields are almost same as the existing solver of TORIC within relative error 0.1%. The agreement of the result indicates the right implementation of the algorithm and good MPI communications between groups.

However, I am still debugging the solver using the full 3D grid because it makes weird small number in the results. The figure 9 represents a part of the result in step 2 of the algorithm, involved in same group, produced by same code using the grid (7,1,1) and grid (7,2,1). The first one (a) is the case making good result whereas the second one (b) is showing underflow in the result. “iam 6” in (a) and “ iam 12” in (b) are located in a same calculation position of the seventh group having same blocks size 39. The fact that Gk are same for both case indicates that there is no error until step 1. Although Gk is same for the both case, T3 become smaller and smaller very rapidly as the redistribution is going (e.g. T3k from o(10^-26) to o(10^-45), when k from 1 to 3) in case of (b) when using the full grid (7,2,1) by the equation T3’=-T3*Gk in step 2. This underflow may be caused by just misuse of Blacs routine when I add more 2 dimension to the matrix operation.

```
iam 6 nblocks 39 k 1
iam 6 T2 (1.,-5.103636957460225E-18) (2
iam 6 T3 (-3.8667903524217541E-17,-1.97
iam 6 Gk (1.903250903449749 79E-14,-2.04
iam 6 ycyc2 (0.E+0,0.E+0) (0.E+0,0.E+0)
iam 6 nblocks 39 k 2
iam 6 T2 (1.,-5.5044323532604 2013E-18)
iam 6 T3 (-2.5994013359663 1372E-17,-1.3
iam 6 Gk (1.9466883571289 3575E-14,-1.24
iam 6 ycyc2 (0.E+0,0.E+0) (0.E+0,0.E+0)
iam 6 nblocks 39 k 3
iam 6 T2 (1.,-5.5145291665692 8352E-18)
iam 6 T3 (-1.9554245906246 9469E-17,-1.1
iam 6 Gk (2.0095594698550 1008E-14,-1.76
iam 6 ycyc2 (0.E+0,0.E+0) (0.E+0,0.E+0)
```

```
iam 12 nblocks 39 k 1
iam 12 T2 (1.,-5.10363695761214236E-18)
iam 12 T3 (-1.52462244424446753E-26,1.6
iam 12 Gk (1.90325090344974979E-14,-2.0
iam 12 ycyc2 (0.E+0,0.E+0) (0.E+0,0.E+0
iam 12 nblocks 39 k 2
iam 12 T2 (1.,-5.10363695765469848E-18)
iam 12 T3 (3.12744885496084 49E-38,2.778
iam 12 Gk (1.94668835712893165E-14,-1.2
iam 12 ycyc2 (0.E+0,0.E+0) (0.E+0,0.E+0
iam 12 nblocks 39 k 3
iam 12 T2 (1.,-5.10363695765469848E-18)
iam 12 T3 (3.68878393031027949E-45,1.51
iam 12 Gk (2.00955946985501513E-14,-1.7
```

Figure9(a)(b). A part of result in step 2 with the processor grid of (7,1,1) (a) and (7,2,1).

4.2. Computation speed of the code

The computation speed of new solver is well evaluated when using only $\boldsymbol{P_1}$ in 3D. It is compared with the existing solver of TORIC using a poloidal 2D grid corresponding to $\boldsymbol{P_2}\,\boldsymbol{P_3}$ in 3D grid. The result is very encouraging, because the new algorithm was turned out almost two times faster than the existing one in normal running condition of TORIC. While the time to complete a running is 881 second for the existing solver with 64 processors, it is 568 second for new solver with 63 processor(Due to the algorithm, it used 2^6-1).

Furthermore, the new solver has another good characteristic showing retardation of the saturation point for the computation speed improvement by increased processors. Because both cases are not a full 3D grid, they cannot help declining the improvement of computation speed as the number of processors increased to the saturation point. In the figure 10, the current solver shows the somewhat flat slope when the number of processors becomes 128, whereas the new solver represents the

continuously deceasing slope, even though the slope is decreased substantially. In the largest number of available processors in Loki cluster, 256, the speed of the new solver is faster more than two times than old solver. The completion time with new solver is 363 sec, whereas it is 855 sec with the old solver. This speed of new solver is at the level satisfying the original purpose of this project.

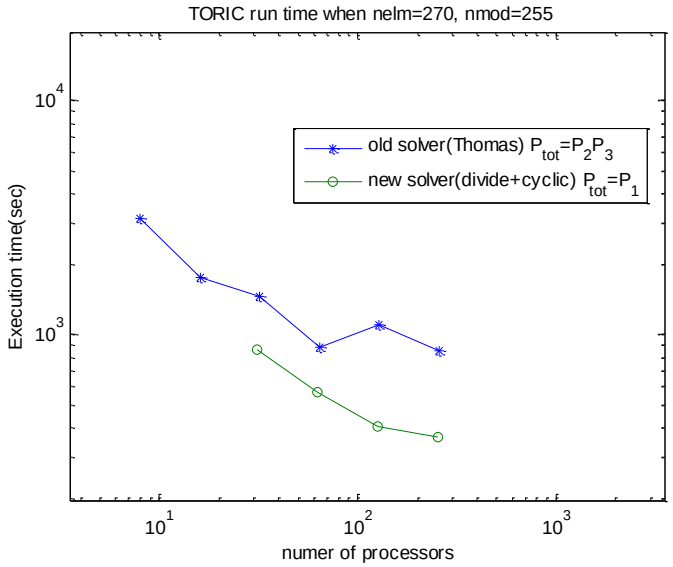


Figure10. The comparison of a run time between old solver and new solver in terms of the number of processors.

4.3. Stability and accuracy of the results

The representing number of the TORIC result is a power absorption by plasma. This value used in normalization of the full electric fields results. Using the new solver, I obtained an average value, 8.533 MW/KA^2 which is close to the result of the old solver within 0.1%. One good point of the new solver is better stability of the result in terms of the number of processors. As shown in figure 11, a variance of the result by the old solver is about 50 times of that by the new solver. This precision may come from the characteristic of the new algorithm. Because the sequential reductions in step 1 are executed in divided groups, the accumulated error can be smaller than that of the old solver which does the sequential reduction for all range of radial components by Thomas algorithm. This characteristic would be very beneficial to calculate the sensitive figures in the result such as induced current drive by the wave in TORIC.

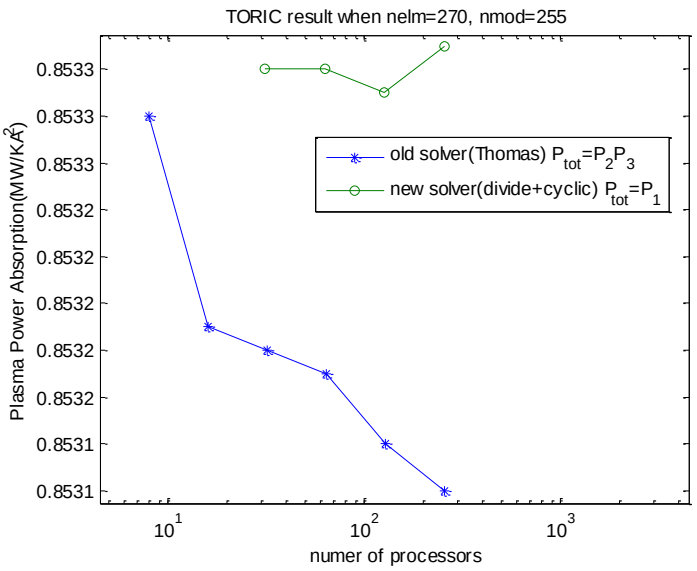


Figure11. The comparison of power absorption between by plama in old solver and new solver of TORIC in terms of the number of processors.

5 Conclusions and Future works

In this project, I made a new solver of TORIC for the parallelization of Block Tri-diagonal system. This solver uses the algorithm suggested by Garaud in the reference 6 which seems like a combination of the divide-and-conquer method and odd-even cyclic reduction. The new solver yields very satisfying results with fast computation speed and good stability as I mention in section 4. When benchmarking with the current solver parallelizing only poloidal components and using serial Thomas algorithm for radial components, the new solver showing about two time faster computation speed and about 50 times smaller variance of the result.

However, I am still developing the new solver to use the full 3-D processor grid instead of 1 dimension parallelization of radial components among Blacs 3-D grids. The use of full 3-D processor grid including both radial and poloidal parallelization would be a breakthrough of the saturation for the improvement of the computation speed by many processors. So, in full 3-D grid, if I use more than 256 processors for TORIC with $\mathrm{N}_{\psi} = 270,\ \mathrm{N}_{\mathrm{m}} = 255$, the completion time can be expected to be less than 5 minutes. Also, it needs to do careful comparisons to decide the value of $\boldsymbol{P_1}$ among total processors by the actual computation speed instead of estimation by the theories as I did in section 2. Finally, this optimization for the ratio of the 3-D processor grid, $(\boldsymbol{P_1}, \boldsymbol{P_2}, \boldsymbol{P_3})$, should be applied in new solver to minimize computation time.

<Acknowledgement>

For this progress report, Prof. Alan Edelman suggested a good guideline for me to complete it. He taught me many parallelization concepts in computation and gave me detailed feedback for the project such as the use of odd-even cyclic algorithm. Two Scientist in PSFC, Dr. Paul Bonoli and Dr. John Wright, participated in this project as advisors. Especially, Dr. Wright suggested this topic for me, and he gave me a lot of practical advice from the experience obtained when he made the current parallel version of TORIC. We are in cooperation to implement the block-tridiagonal solver in TORIC, and we hope that the solver will be useful for the future research on analysis of interaction between plasma and electromagnetic wave in TOKAMAK.