PSFC/JA-10-9

A versatile parallel block-tridiagonal solver for spectral codes

Lee, J., Wright, J.

May 2010

Plasma Science and Fusion Center
Massachusetts Institute of Technology
Cambridge MA 02139 USA

This work was supported by the U.S. Department of Energy, Grant No. DE-FC02-01ER54648.

A versatile parallel block-tridiagonal solver for spectral codes

Jungpyo Lee and John C. Wright

Abstract:

Three-dimensional (3-D) processor configuration of a parallel solver is introduced to solve a massive block-tridiagonal matrix system in this paper. The purpose of the added parallelization dimension is to retard the saturation of the scaling due to communication overhead and an inefficient parallelization. The semi-empirical formula for the matrix operation count of the typical parallel algorithms is estimated including the saturation effect in 3-D processor grid. As the most suitable algorithm, the combined method of "Divide-and-Conquer" and "Cyclic Odd-Even Reduction" is implemented in a MPI-Fortran90 based numerical code named TORIC. The new 3-D parallel solver of TORIC using thousands of processors shows about 4 times improved computation speed at the optimized 3-D grid than the old 2-D parallel solver in the same condition.

1. Introduction

In the solution of partial differential equations in two dimensions, a block-tridiagonal matrix system may appear. Typically, in along one coordinate only adjacent mesh points or elements are coupled by the discretization resulting in the tridiagonal structure. If the coupling along the second dimension is in a local basis, the resulting blocks are small compared to the number of block rows. When a global basis is used (for example a Fourier basis), the size of the blocks may be comparable to the number of block rows. For larger sized problems, in core memory may not be sufficient to hold even a few blocks, and so the blocks must be distributed across several cores. Many methods have parallelized the system by considering the system as just either "tridiagonal" system[1-4] or "block" matrix system[5]. The "tridiagonal" system is parallelized in the block row dimension, whereas "block" matrix system is parallelized by distributing the blocks. In other words, sometimes people adopted the parallelized algorithm for block tridiagonal system in which the rows of the master matrix are divided among a one-dimensional processor grid and they are calculated with "cyclic reduction method" [1] or "divide-and-conquer" [2,3] as in the simple tridiagonal problem. The other way to parallelize the system is to keep the serial routine of Thomas algorithm[6], and, for each block operation, use a parallelized matrix computation algorithm such as ScaLAPACK[7] in a two-dimensional (2-D) processor grid. However, both parallelization methods have limitations for scaling to a large number of processors. To overcome this and achieve better scaling, we combine both parallelization methods in a system using three-dimensional (3-D) processor grid.

TORIC[8] is a MPI-Fortran90 based numerical code which has been used to investigate the interaction between plasmas and driven electromagnetic wave in the toroidal geometries of tokamaks[9]. The three coordinates of toroidal geometry are radial (ψ) , poloidal (θ) and toroidal(ϕ) direction. TORIC is three-dimensional linear system using the finite element method (FEM) with cubic Hermite polynomial basis in radial direction and Fourier spectral analysis in poloidal direction and toroidal directions. The toroidal direction is taken to be axi-symmetric so there is no coupling in that direction and the system is reduced to a set of 2D problems each parameterized by a toroidal mode number. The solver of TORIC computes a block-tridiagonal matrix problem and the equations in the solver can be expressed as Eqn. (1)

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

Each $\vec{x}_i$ is a complex vector of $6N_m$ poloidal Fourier components, and the size of the three blocks, $\underline{L_i}$,$\underline{D_i}$ and $\underline{R_i}$ is $(6N_m) \times (6N_m)$. The number of radial elements, Npsi, determines the number of block rows. For simplicity, let $n_1 = N_\psi$ and $n_2 = 6N_m$. Then, the total master matrix size is $(n_1 n_2) \times (n_1 n_2)$, with typical values of n_1 and n_2 for a large problem being about 1000 and 6000 (See Fig. 1).

The current parallel solver in TORIC was implemented with the serial Thomas algorithm along the block rows and 2-D parallel operations for blocks.[6] ScaLAPACK (using routines: `PZGEMM`, `PZGEADD`, `PZGETRS`, `PZGETRF`)[7] was used for all matrix operations including the generalized matrix algebra of the Thomas algorithm. When the square of the number of the poloidal modes (N_m^2) is much larger than the number of processors (P_{tot}), this implementation is very efficient. The logical block size used by ScaLAPACK[7] is set to 72x72 and so P_{tot} is constrained to be less than $(6N_m/72)^2$. A smaller logical block size may be chosen to increase the number of processors available for use at the cost of increased communication. We have found experimentally that communication degrades performance even as this constraint is approached. The shortest possible total run time is needed to run TORIC several times in a big synthetic code for plasma analysis in a TOKAMAK. TORIC is used as a physics component in two integrated modeling efforts in the fusion community, CSWIM[10] and Transp[11]. The current small limitation for the number of processors may present load balancing problems and induce many free processors in a large multi-component coupled. For a relatively small problem, $N_\psi = 270$, $N_m = 255$ with 20 processors, the completion time to run TORIC is about one hour. Our purpose is to reduce the time to an order of minutes by the use of about 1000 processors by better scaling. In this sense, we need to make a parallelization of the radial direction as well as the poloidal direction for the better scaling of the block-tridiagonal system.

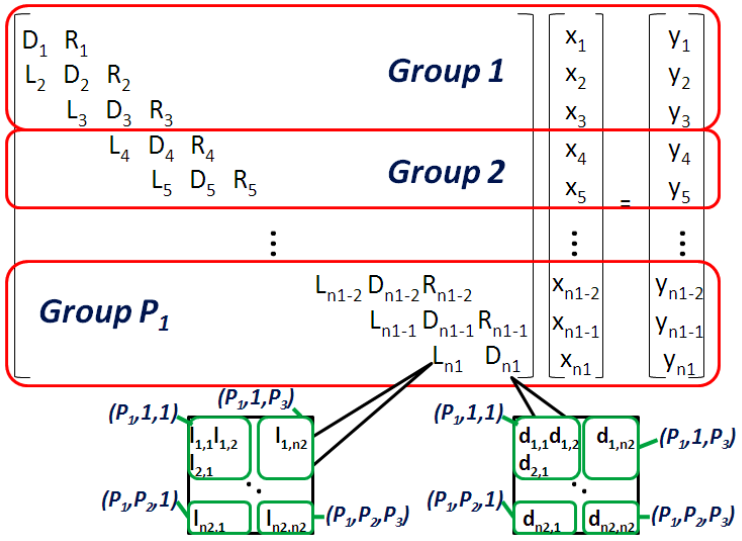

Figure 1. A schematic of the 3-dimensional parallelization for a block-tridiagonal system. The size of each block, L, D and R is $n_2 \times n_2$,and there are n_1 rows of the blocks. The rows are divided by P_1 groups, and the element of each block is assigned to $P_2 P_3$ processors. So, every element has a 3-dimensional index of the assigned processor among total number of processors, $P_{tot} = P_1 P_2 P_3$.

2. Selection of the parallel algorithm

To select a parallel algorithm adequate for the new solver, we compared the matrix operation count and the required memory for the algorithms typically used for block-tridiagonal matrix solver in Table 1. The common parallel block-tridiagonal solvers use a 1-D processor grid, (i.e. $P_{tot} = P_1$) because the algorithm, whether it is divide-and-conquer or odd-even cyclic reduction, is easily applicable to a 1-D processor grid, and the usual size of blocks, n_2, is much smaller than the number

of blocks, n_1. However, in our case for TORIC, usually, n_2 is as big as n_1, so we require parallelization of each matrix block as well as of the tridiagonal system because of the required memory and desired calculation speed. Saving several blocks each of size $n_2 \times n_2$ in a core would be impossible. It is for this reason that the data in each block is distributed on a 2-D processor grid in the current version of TORIC solver.

If the block operation time is ideally reduced by the number of processor used in the operation, it is always the most efficient in terms of number of floating point operations and memory to use Thomas algorithm using the serial calculation in rows and parallelized block operations in 2-D processor grid (i.e. $P_{tot} = P_2 P_3$) as the current parallel solver of TORIC. However, the additional operation for the parallelization and the increased communication between the processors deteriorate the speed improvement by parallelization as the number of processors increases and becomes comparable to the the square root of the size of a block divided by the logical block size [e.g. $P_{tot} \sim (6N_m/72)^2$]. Also, beyond this limit, additional processors have no work to do and remain idle. A good way to avoid both memory and speed problems and retain full utilization of processors is to add another dimension for the parallelization, so the total processor grid configuration becomes three dimensional (i.e. $P_{tot} = P_1 P_2 P_3$.)

While the cyclic odd-even reduction algorithm has no fill-in matrix which requires quite a long additional time in the divided-and-conquer algorithm, it has more matrix operations in a row and many processors are free during the reduction process. Additionally, the cyclic reduction algorithm has a constraint that both P_1 and n_1 should be about a power of 2. When P_1 is assigned to be less than $\frac{n_1}{2}$, it induces $(\frac{n_1}{2P_1} - 1)$ additional serial process in the cyclic reduction algorithm operation count. It may depreciate the advantage of the logarithmic reduction considering the relatively high matrix operation count (See Table 1).

The combined algorithm of the cyclic reduction and divided-and-conquer algorithm was introduced in the reference 8. They used this combined algorithm for the analysis on solar tachocline to enhance both the speed and the stability of the calculation[8]. It can alleviate the local pivoting instability problem of the cyclic reduction method because it is based on the divide-and-conquer method except that it uses the cyclic reduction method for dealing with fill-in matrix and the communication between P_1 groups. So, n_1 doesn't have to be a power of 2 and it can save the time for reducing fill-in matrix. The matrix operation count for the fill-in reduction in divide-and-conquer method, $P_1(M + A + 2D)$, is replaced by the term from the cyclic reduction algorithm in the combined algorithm, $(\log_2 P_1 - 1) \times (13M + 6A + D)$ (See table 1). Even in the combined algorithm, P_1 should be about power of 2.

We made a elapsed time model for the matrix parallel operation, multiplication M, addition A, and division D including saturation effect in Eq.(2)-(4) to compare the realistic speed of the algorithms. From the observation that the deterioration of scaling by parallelization become severe as the number of processors approaches the saturation point, we set the exponential model in Eq. (5).

$$M = M_0 \frac{n_2^2}{(P_2 P_3)_{eff}} \tag{2}$$

$$A = A_0 \frac{n_2^2}{(P_2 P_3)_{eff}} \tag{3}$$

$$D = D_0 \frac{n_2^2}{(P_2 P_3)_{eff}} \tag{4}$$

$$(P_2P_3)_{eff} = \left\{(P_2P_3)_{sat} * \left(1 - \exp\left(-\frac{(P_2P_3)}{(P_2P_3)_{sat}}\right)\right)\right\}^{\alpha_{(P_2P_3)}} \quad (5)$$

$$(P_1)_{eff} = \left\{(P_1)_{sat} * \left(1 - \exp\left(-\frac{(P_1)}{(P_1)_{sat}}\right)\right)\right\}^{\alpha_{(P_1)}} \quad (6)$$

The exponent parameter, $\alpha_{(P_2P_3)}$, represents the non-ideal scaling because $(P_2P_3)_{eff}$ becomes about $(P_2P_3)^{\alpha_{(P_2P_3)}}$ when P_2P_3 is much smaller than $(P_2P_3)_{sat}$. Ideally, $\alpha_{(P_2P_3)}$ should be 1. However, by the real test of the run time in section 4, we can specify the parameters, $\alpha_{(P_2P_3)} = 0.41$ and $(P_2P_3)_{sat} = 64$. Since this saturation effect model can be applied for all parallelized directions, P_1 in table 1 can be replaced with $(P_1)_{eff}$ in Eq. (6), and we can define$(P_1)_{eff} = \frac{n_1}{2}$ and $\alpha_{(P_1)} = 0.84$ by the slope of the run time result. Also, we set the parameters, $M_0 = 0.5D_0 = n_2A_0$, because the general speed of matrix multiplication in a well optimized computation code is about two times faster than that of matrix division by experience when the matrix size is about 1000×1000.

The count operation of the algorithms by our model is shown in the graphs in Fig.2 for relatively small size system problem in TORIC, $n_1 = 270$ and$n_2 = 6 \times 255 = 1530$. The combined algorithm is estimated to have a minimum computation time at a specific 3-D grid configuration. Although we didn't consider the communication time in detail and negligible vector operation time, this model seems to be precise because the real computation result with same size problem shown in Fig. 9 shows very similar pattern with the estimation by the model in Fig.2 (Compare the blue and black curves in Fig. 2 with Fig. 9). Thus, we selected the combined algorithm of divide-and-conquer and cyclic odd-even reduction method for the new solver of TORIC.

	Cyclic Odd-Even Reduction Algorithm[1]	Divide-and-Conquer Algorithm[2]	Combined Algorithm[4]
Elapsed time for matrix operations for a processor	$(\log_2 P_1 + (\frac{n_1}{2P_1} - 1)) \times (13M + 6A + D)$	$\frac{n_1}{P_1}(4M + 2A + 2D) + P_1(M + A + 2D)$	$\frac{n_1}{P_1}(4M + 2A + 2D) + (\log_2 P_1 - 1) \times$
Maximum memory for a processor	$\frac{n_1}{P_1}(2n_2^2 + 2n_2)/P_2P_3 \times complex\ type$	$\frac{n_1}{P_1}(2n_2^2 + 2n_2)/P_2P_3 \times complex\ type$	$\frac{n_1}{P_1}(2n_2^2 + 2n_2)/P_2P_3 \times complex\ type$

Table 1. Comparison of Block-Tridiagonal algorithms with n_1 row of blocks which size is $n_2 \times n_2$. When total number of processors is $P_{tot} = P_1P_2P_3$, n_1 is parallelized in P_1 groups and each block in a group is parallelized according to P_2P_3 processor grid during the matrix(block) operation. So, in a processor, the required time for block operation, multiplication M, addition A, and division D is an order of $\frac{n_2^2}{P_2P_3}$ ideally.

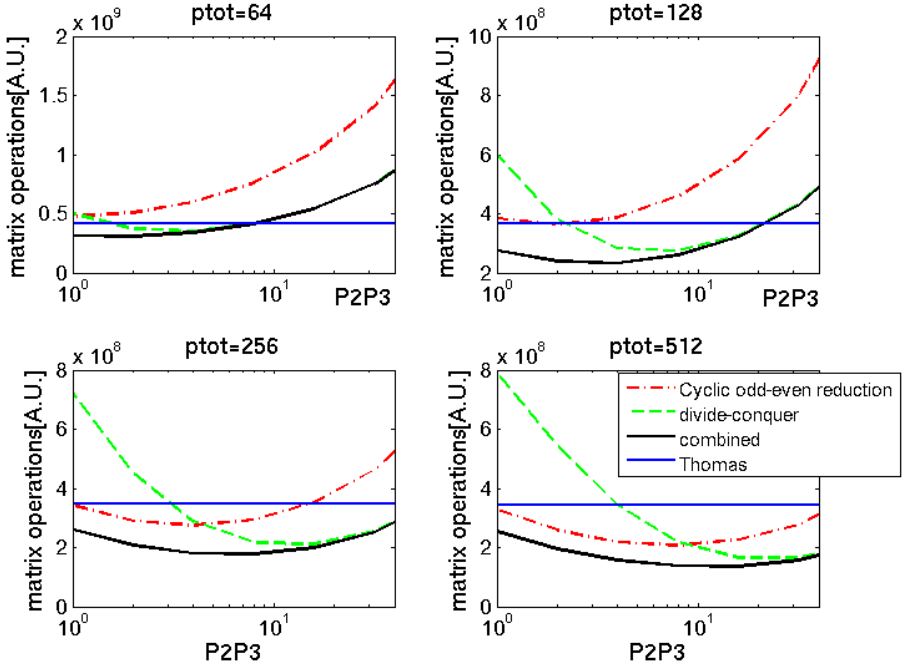

Figure 2. The estimation of the matrix operation count by various parallel algorithms in terms of the 3-D processor grid configuration for the case, $n_1 = 270$ and $n_2 = 1530$. It is based on the table 1, and the saturation effects are included by the model in Eq.(2)-(6) for M, A, D and P_1 .

3. Code Implementations

3.1. Set up a 3-dimensional processor grid

One way to implement 3-D grid is to use a context array in BLACS[13], in which each context has uses 2-D processors grid as it is already implemented in the current solver. In BLACS, a context indicates a group within a boundary of an MPI communicator. In the default context having the total number of processors, it is possible to assign several sub-groups of processors corresponding to each context according to the specific maps (see an example in Fig. 3) The sub-groups are able to communicate with each other when needed in a tridiagonal algorithm.

```
*     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)
```

Figure 3. An example for the multi context of 3-D processor grid in BLACS. $[P_1, P_2, P_3] = (2,2,2)$

3.2. Divided forward elimination and Odd-even cyclic reductions

The combined algorithms of divide-and-conquer method and cyclic odd-even reduction[4] can be summarized as the three forward reduction steps and two back substitution steps. The three forward steps are described in Fig. 4. The first step is for the serial elimination of "L" block by the previous row as in the Thomas algorithm, but this process is executed simultaneously in every group like a typical divide-and-conquer method. During the first step, the elimination processes create redundant fill-in blocks "F" at the last non-zero column of the previous group except the first group (See Fig. 4).

Step 2 is the preliminary process for the step 3, the cyclic reduction step which requires tridiagonal form. To make the form composed of the circled blocks in the first row of each group, we need to move the blocks "G" in the first row to the column where the block "E" of the next group is located. Before carrying out the redistribution, the matrices in last row in each group should be transmitted to the next group. Then, the received right block "R'" is eliminated by the appropriate linear operation with a following row, and the elimination by the next row is repeated until the block "G" moves to the position.

The redistributed tri-diagonal forms can be reduced by a typical odd-even cyclic reduction in step 3 as shown in Fig. 4. This step is for the communication of information between groups, so the portion of the total run time for this step is increased as P_1 is increased. This reduction is carried out in $\log_2 \frac{(P_1+1)}{2}$ steps because P_1 should be $2^n - 1$ instead of 2^n where n is an integer, and it requires the total number of processors to be several times $2^n - 1$. 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 small number of free processors in certain nodes always.

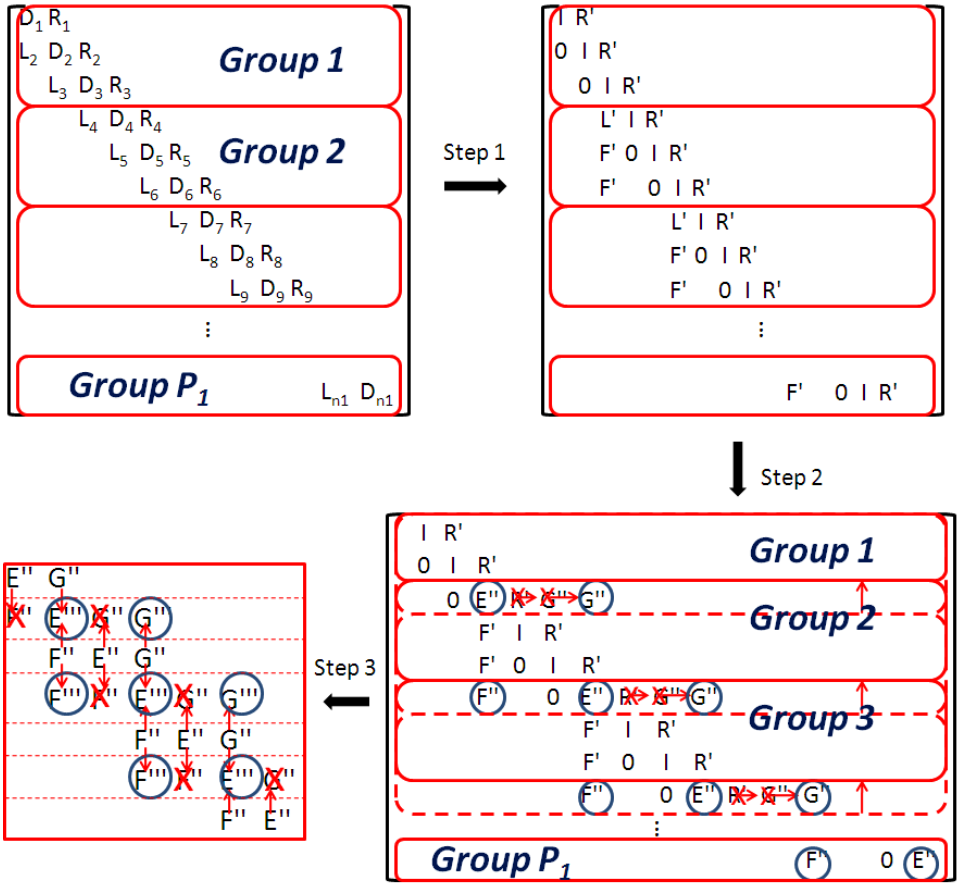

Figure 4. The code description for the forward elimination (Step 1), the block redistribution (Step 2), and Cyclic odd-even reduction (Step 3).

3.3. Cyclic substitutions and divided backward substitutions

In the end of the cyclic reduction in step 3, only one block "E" remains, so we can obtain a part of the solution at last by $X_i = \mathrm{E}^{-1} y_i$. The part of the solution is substituted to find a full solution vector "x" in step 4 and step 5. In step 4, the cyclic back substitution is executed in $\log_2 \frac{(P_1+1)}{2}$ steps, the same as in the cyclic reduction step. Then, in each group, the serial back substitution continues simultaneously in step 5. In this step, each group except the first one should have the information of the solution in the previous group to evaluate the terms contributed by the fill-in blocks "F" in the solution.

4. Result and Discussions

4.1. Computation speed of the code

The computation speed of the new solver of TORIC is evaluated with various 3-D processor grid, $[P_1, P_2, P_3]$, for three different size problem, $[n_1, n_2] = (270,1530)$ in Fig.5, $(480,3066)$ in Fig. 6 and $(980,6138)$ in Fig.7. The evaluation is conducted on the Franklin cluster in National Energy Research Scientific Computing Center (NERSC). Each of Franklin's compute nodes consists of a 2.3 GHz quad-core AMD Opteron processor (Budapest) with a theoretical peak performance of 9.2 GFlop/sec per core. Each core has 2GB of memory. The result by the new solver is compared with that of the old solver using only 2-D processor grid corresponding to $P_2 P_3$, by Thomas algorithm in the same environment. In the best case, the combined algorithm of the new solver is 4, 6 or 10 times faster than Thomas algorithm of the old solver for small, medium and large size problem respectively as shown in Fig. 5-(c), Fig. 6-(c) and Fig. 7-(c) respectively.

In the log-log graph of the run time as a function of the number of processors, an ideal scaling by the parallelization has a slope of -1. Although the ideal scaling is hard to be obtained because of increasing communication, the new solver shows much steeper slope than the old solver. The new solver has another good characteristic showing retardation of the saturation point for the computation speed improvement by increased processors. The saturation points for P_1 and $P_2 P_3$ used in the model of section 2 can be inferred from the condition when the graph become somewhat flat in Fig. 6-(c). We observed that the slope for the new solver became steeper as $P_2 P_3$ became larger in all size problems in the Fig. 5-(c), 6-(c) and 7-(c). It implies the beneficial effect of the balanced 3-D processor grid distribution even before the saturation point. Also, generally the slope became steeper for the bigger size problem or the smaller processors. Those facts validate the exponential form of the model in section 2.

Since TORIC is an independent computational code, the total run time is significantly influenced by the pre-processing time and post-processing time as well. During the pre-processing, by the plasma physics, it fills meaning complex numbers in each block which is distributed in the processors according to the grid configuration of the solver. So, the reduced pre-processing time is another important asset of 3-D processor grid by the new solver as shown in Fig. 5-(b), Fig. 6-(b) and Fig. 7-(b), even though the pre-processing is not related to the algorithm of the solvers. While the graphs of the 3-D processor grid configuration by the new solver indicate almost the ideal scaling in the graph, the red graphs of the existing solver shows only discrete improvement every quadrupling of P_{tot}. On the other hand, the post-process has no relation with the solvers at all, because it use a different 1-dimensional parallel computation routine by distributing n_1 in total number of processors, P_{tot}. That's why the new solver and the current solver have similar pattern showing ideal scaling when P_{tot} is below n_1, and non-scaling after the saturation point in Fig. 5-(d), Fig. 6-(d) and Fig. 7-(d).

The optimal grid configurations exist at the minimum total run time when $P_2 P_3 = 16$ for the small and medium size problem and $P_2 P_3 = 128$ for the large size problem if the number of processors is big enough. The figure 8 represents the run time comparison in terms of $P_2 P_3$ for the small size problem. As we mentioned in section 2, this result is reasonably consistent with the non-ideal scaling model we developed for the algorithm comparison. We have to notice that the new solver is not always faster than the old solver because Thomas algorithm has smaller matrix operations theoretically and it works fine with the small number of processor far before the saturation point.

Figure 9 shows the allocated computation time for the steps of the combined algorithm within the run time of the new solver. The actual scaling in terms of P_1 for each step is well accordant with theoretical scaling described in the section 2. Since the matrix operation of the divided-and-conquer part in the step 1 and step 2 is proportional to $\frac{n_1}{P_1}$, the slope is about -1. But the communication part between P_1 groups using cyclic reduction in step 3 makes logarithmic increase of the graph because

the matrix operation count is proportional to $\log_2 P_1$, as indicated in Table 1. For large P_1 with a fixed $P_2 P_3$, the run time of step 3 is dominant in the total run time, which imply the saturation of 1-D processor configuration for the new solver.(See also the reduced slope of the yellow line for large P_{tot} in Fig. 5-(c)) Thus, 3-D processor configuration should be used for large P_{tot} to avoid the saturation due to the communication between P_1 groups.

The better speed improvement of step 1 for the large P_1 than ideal scaling in Fig. 9 is derived from an algorithm reason that the matrix operations of the first row in each group is much smaller than the rest rows in the group, and some of the remaining rows become the first rows in a group as we divide more groups by the increased P_1.

From table 2 obtained by IPM which is a monitoring tool in NERSC, we can compare the saturation effect by MPI communication for two solvers. Although the tool measures the communication time not in a specific subroutine but in the total run time including pre-processing and post-processing, we can see the remarkable difference between the old solver and the new solver. MPI communication time increase in terms of the total core number for the old solver is much faster than the new solver. Also, the drop of the floating point operation speed (Gflop/s) in terms of the total core number for the old solver is much severe than the speed drop for the new solver. The both facts demonstrate the retarded saturation for the new solver by reduced communication. We can see also about 3-5 times higher average operation speed for the new solver than that of the old solver. It may be due to not only less communication overhead but also more efficient data processing in the new solver algorithm.

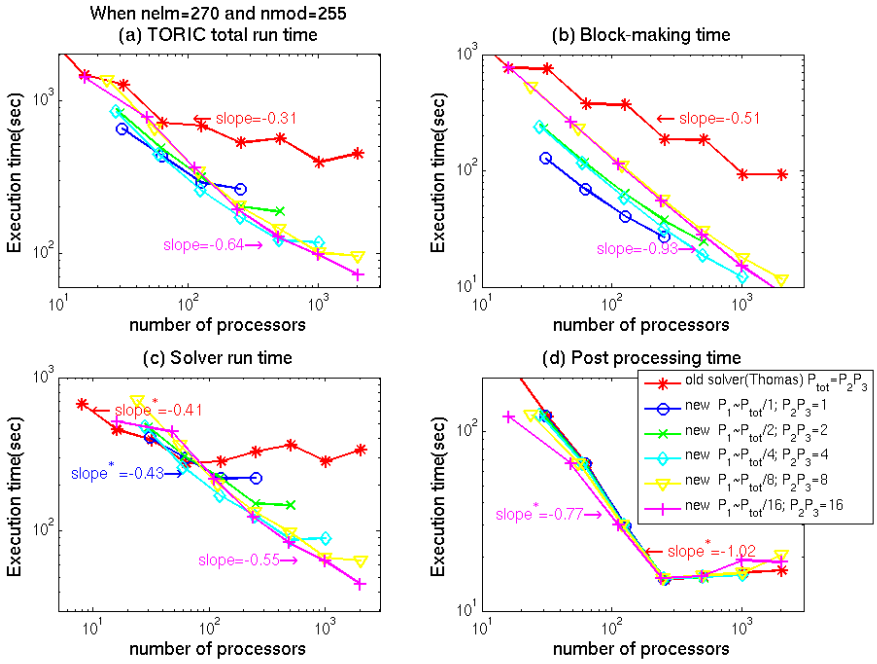

Figure 5. The total run time(a), pre-processing time(b), solver time(c) and post-processing time(d) for the old solver and the new solver of TORIC in terms of various 3-D processor grid configuration $[P_1, P_2, P_3]$ with a small size problem $[n_1, n_2] = (270,1530)$. The total run time (a) is a sum of the other times of (b), (c) and (d).

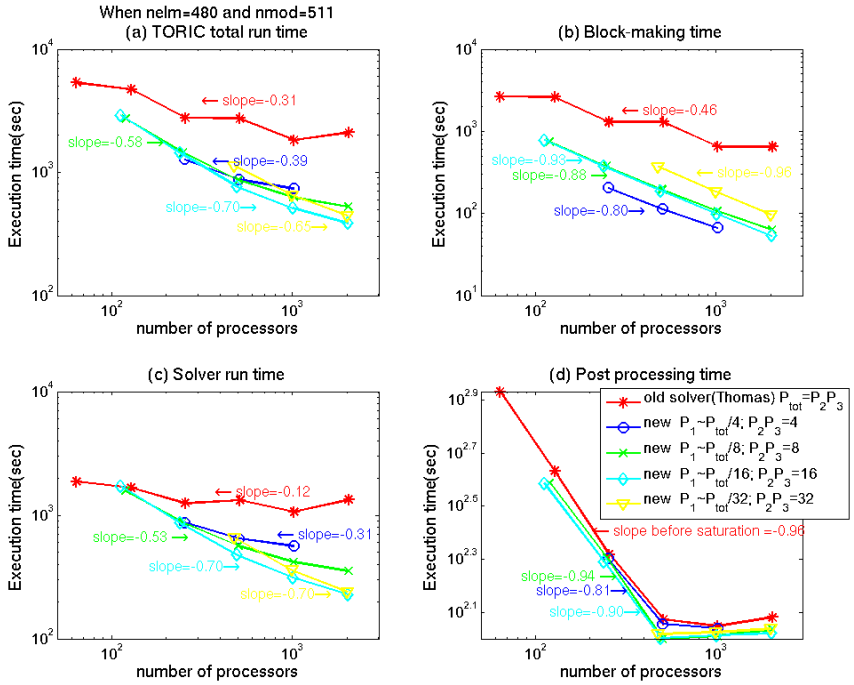

Figure 6. The total run time(a), pre-processing time(b), solver time(c) and post-processing time(d) for the old solver and the new solver of TORIC in terms of various 3-D processor grid configuration $[P_1, P_2, P_3]$ with a medium size problem $[n_1, n_2] = (480,3066)$.

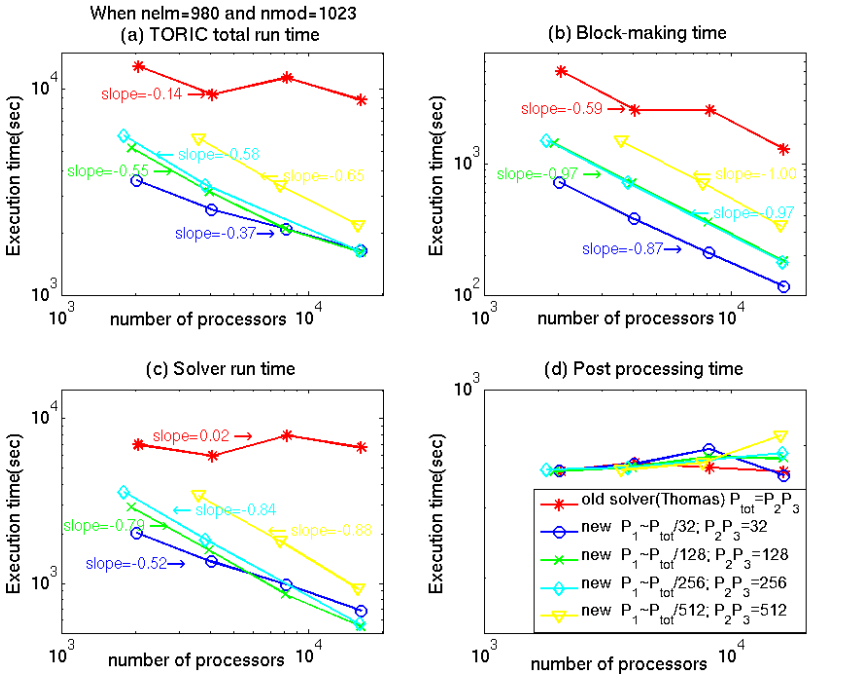

Figure 7. The total run time(a), pre-processing time(b), solver time(c) and post-processing time(d) for the old solver and the new solver of TORIC in terms of various 3-D processor grid configuration $[P_1, P_2, P_3]$ with a large size problem $[n_1, n_2] = (980,6138)$.

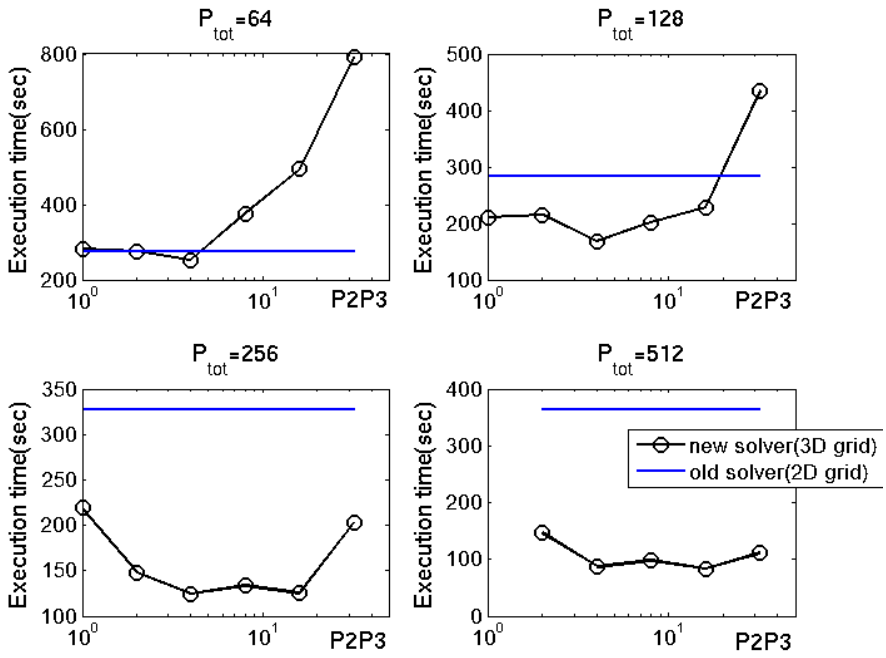

Figure 8. The comparison of the solver run time in terms of P_2P_3 with a small size problem $[n_1, n_2] = (270,1530)$. Compare this result with the estimation for the combined algorithm (black) and Thomas algorithm (blue) in Fig. 2 by our saturation model.

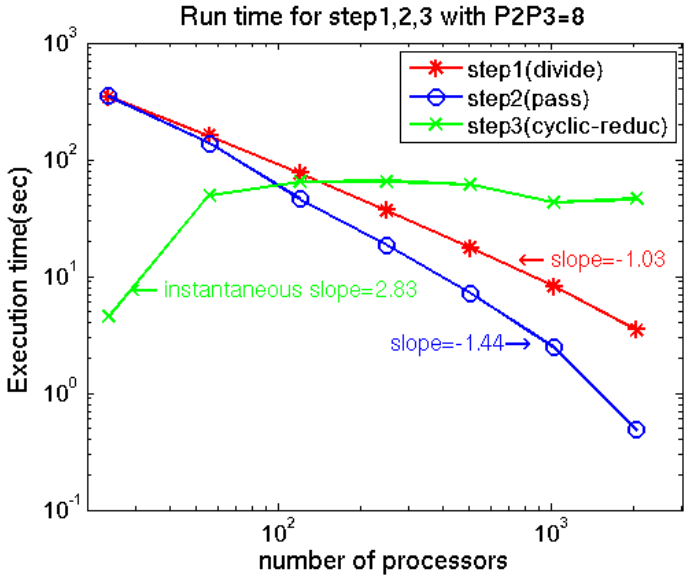

Figure 9. The run time scaling of the forward reduction steps in the new solver with a small size problem $[n_1, n_2] = (270,1530)$. Step 1 is a divided forward elimination process. Step 2 is a preliminary process for making the tridiagonal form needed for Step 3, which is a typical cyclic odd-even reduction process. The step 3 shows the logarithmic increase as indicated in Table 1. The summation of the three run times of step 1,2, and 3 is corresponding to the yellow graph in Fig. 5-(c)

	%comm(avg)	gflop/sec(avg)	gbyte(avg)
Old solver(ptot=32)	26.1933	0.719465	0.522082
Old Solver(ptot=128)	38.4359	0.437786	0.291693
Old Solver(ptot=2048)	78.0572	0.109938	0.188168
New solver (ptot=32,p2p3=1)	34.6192	2.59628	1.48556
New Solver (ptot=128,p2p3=1)	53.5348	1.71394	1.05067
New Solver (ptot=128,p2p3=16)	48.6689	1.15822	0.391859
New Solver (ptot=2048,p2p3=16)	64.2425	0.567634	0.262266

Table 2. The average MPI communication time percentage (The first column), the floating point operation speed (The second column), and the average memory usage per a core (The third column) measured by IPM which is the NERSC developed performance monitoring tool for MPI program. This result is for the small size problem $[n_1, n_2] = (270,1530)$ in terms of various processor grid configuration and solver types (See Fig 5 (a) for the total run time of the same problem size)

4.2. Other issues of the code

The required memory in a core for the new solver is about two times of that for the old solver because of the fill-ins blocks (See Table 1 and 2). For 16 processors, the allocated memory per core is about 2GB by the new solver, so it restrains us from testing the new solver with processors less than 16 and the old solver below 8 processors (See Fig. 5). An out-of-core method would enable the new solver to work with small number of processors, but the calculation speed would be decreased.

For accuracy of the new solver, we can compare a wave power absorption value that is calculated in the post-processing and is based on the full solution. This value is used in normalization of the full electric fields results. Using the new solver, we obtain an average value, 8.533 MW/KA^2 which is close to the result of the old solver within 0.01%.

Also, the new solver shows good stability of the result in terms of the variance of processor number within 0.01%. This good precision may come from the characteristic of the new algorithm. Because the sequential eliminations in step 1 are executed in divided groups, the accumulated error can be smaller than that of the old solver which does the sequential elimination for all range of radial components by Thomas algorithm. However, from another viewpoint, the local pivoting in the divided groups of the new solver instead of the global pivoting in Thomas algorithm may induce instability of the solution. Many people have investigated the relevant stability of the tridiagonal system with divided-and-conquer algorithm[14] and cyclic reduction algorithm[15] and have developed a technique to assure the numerical stability regarding the pivoting[16,17].

Because the speed of the solver is determined by the slowest processor, a well distributed load over all processors is very important. In Fig. 10, the most unbalanced of the work load occur during pre-processing because the blocks made in TORIC for the edge of the physical domain are trivial such as an identity or zero matrix for the last several processors. All processor are blocked by MPI barrier until they reach the backward substitution step, so the last several processors usually are free at the end of the runtime for step 3. We may use this expected unbalance by assigning more work during the solver time for the free processors. Then, the unbalance would be dissolved after step 1 and make the new solver faster.

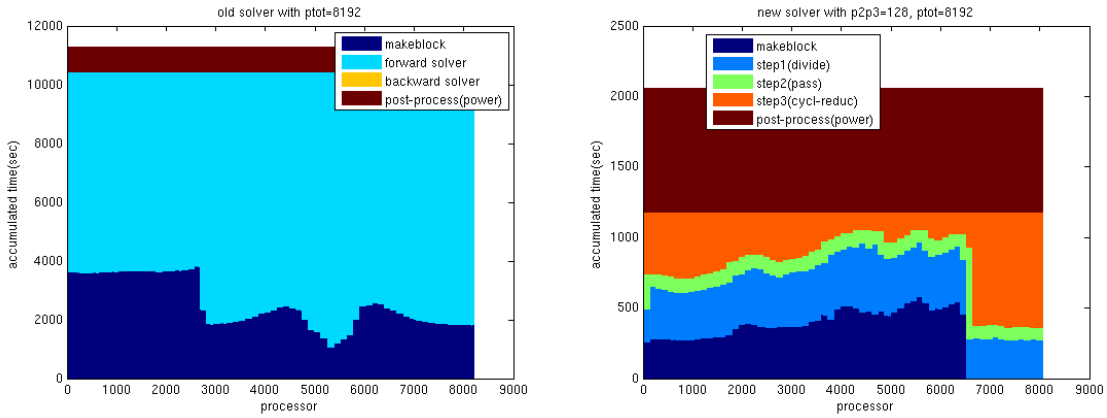

Figure 10-(a)(b). Work load distribution to each processor when $P_{tot} = 8192$ and $[n_1, n_2] = (980,6138)$. X axis indicates a processors index, and y axis means the accumulation run time from the beginning of the old solver (a) and the new solver (b).

5. Conclusion

The optimized distribution of total processors in 3-D configuration for massive Block-tridiagonal system is shown to be beneficial for faster computation by reducing the communication overhead when large number of processors is given. Although a 3-D solver using the combined method of "Divide-and-Conquer" and "Cyclic Odd-Even Reduction" requires about double size memory than a 2-D solver using "Thomas algorithm", it shows much bigger floating point operation rate, and good accuracy and stability of the solution.

Acknowledgments

We would like to thank P. Garaud for supplying the source code of his 1D block-tridiagonal solver using the combined method. This work is supported in part by a Samsung Fellowship, USDoE awards DE-FC02-99ER54512 and the DOE Wave–Particle SciDAC (Scientific Discovery through Advanced Computing) Contract No. DE-FC02-01ER54648. This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

Reference

1. H.S.Stone, ACM transactions on Mathematical Software,Vol1(1975),289-307
2. H.H.Wang, ACM transactions on Mathematical Software,Vol7(1981),170-183
3. V.Mehrmann,Parallel Computing 19(1993),257-279
4. P.Garaud, Mon.Not.R.Astron.Soc,391(2008)1239-1258
5. J.C.Wright, P.T. Bonoli & E.D.B. Azevedo, *Computer Phys. Comm*,*164*(2004) 330-335
6. L.H.Thomas, Elliptic problems in linear difference equations over a network, Watson Sci. Comput. Lab. Rept., Columbia University, New York, (1949).
7. ScaLAPACK Users' Guide, available from SIAM, ISBN 0-89871-397-8 (1997)
8. M.Brambilla, Plasma Phys.Control.Fusion 41(1999) I-34
9. N.J. Fisch, Rev.Mod.Phys,59(1987),175-234
10. D. Batchelor, et al, "Advances in simulation of wave interations with extended MHD phenomena," in H. Simon, Ed.,"SciDAC 2009, 14-18 June 2009, California, USA", v. 180 of Journal of Physics: Conference Series , page 012054, Institute of Physics, (2009).
11. Hawryluk, R.I. "An Empirical Approach to Tokamak Transport", Course on Physics of Plasma Close to Thermonuclear Conditions, ed. by B. Coppi, et al., (CEC, Brussels, 1980), Vol. 1, pp. 19-46.
12. S.D. Conte, and C. deBoor, "Elementary Numerical Analysis", (1972), McGraw-Hill, New York.
13. BLACS Users' Guide, available from http://www.netlib.org/blacs/lawn94.ps
14. V.Pavlov and D.Todorova, WNAA(1996), 380-387
15. P.Yalamov and V.Pavlov, Linear Algebra and its applications 249(1996),341-358
16. N. J. Highham, Linear Algebra and its applications 287(1999),181-189
17. G. Alaghband, Parallel Computing 11(1989) 201-221