OpenMP/MPI Hybrid Parallel ILU(k) Preconditioner for FEM Based on Extended Hierarchical Interface Decomposition for Multi-core Clusters

Masae Hayashi and Kengo Nakajima

Information Technology Center, The University of Tokyo, 2-11-16 Yayoi Bunkyo-ku
Tokyo, Japan

Abstract. While ILU preconditioner is a powerful and popular preconditioning method for Krylov iterative solvers on sparse matrices derived from finite element analysis, it have been exploerd the scalable hybrid parallelization scheme for ILU preconditioner targetting multi/many-core clusters. Hierarchical Interface Decompostion (HID) is a robust and efficient parallel method for ILU preconditioner. The extended version of HID (ExHID), our proposed method, introduces thicker level-2 connector in order to consider fill-ins. Basing on HID and ExHID we developed hybrid parallel ILU preconditioner with fill-ins using OpenMP/MPI hybrid parallel programing models. While inter-node parallelization is based on HID/ExHID, we applied two different methods, multicolor based reordering and HID/ExHID to intra-node parallelization. The two implementations according to different hybrid strategy, HID(inter-node)-HID (intra-node) and HID(inter-node)-MC(intra-node), are evaluated through strong scaling tests and the better hybrid strategy is explored. HID-HID generally results with better convergence and less fill-ins. On the other hand, HID-MC could be more stable strategy than HID-HID when increasing the number of threads per process.

1 Introduction

Domain decomposition method(DDM) is widely used parallelization method in many finite element applications. Then distributed sparse linear systems derived from each subdomain is considered as distributed objects[1,2] on each process. On the other hand, preconditioining method using incomplete LU factorization without fill-in (ILU(0)) is popular and effective preconditioner for finite element applications. Under the parallelization based on DDM, block Jacobi-type localized preconditioner are widely used for parallel iterative solvers[3,4]. While they provide excellent parallel performance for well-defined problems, the number of iterations for convergence increases gradually according to the number of processors. Moreover this preconditioner decreases its robustness for ill-conditioned problems with many processors, since it ignores the global effect of external nodes come of inherently sequential natures of ILU preconditioner. The common remedy is to extend the overlapped elements between domains[5,6]. At the expenses of additional computation and communications it still only allows us to consider the global effect without updates from previous row operations. Another common remedy to parallelize ILU preconditioner is multicoloring based ordering. Multicoloring the subdomains which is assigned to processors, yeilds the parallelism from a global ordering[7]. But the archieved parallelism is limited to the color number and fiding optimal color number becomes another difficulty which relates to the performance. The Parallel Hierarchical Interface Decompostion Algorithm (PHIDAL) provides robustness and scalability for parallel ILU/IC preconditioners basing on “hierarchical interface decompostion (HID)“[8]. HID exploits a hierarchical decompostion of the graph which yields natural parallelism in the factorization process. For taking into account fill-ins, we introduced additional layers in higher level connectors defined in HID. The proposed method is called Extended HID (ExHID). We developed parallel ILU preconditioners with fill-ins (ILU(k)) basing on HID/ExHID.

To enhance our parallel ILU(k) preconditioner for multi-core environment, we applied a hybrid parallel programming model. A hybrid parallel programming model is often employed in order to archieve minimal parallelization overheads on multi-core clusters. Corse-grained parallelism is archieved through domain decompostion by message passing among nodes, while fine-grained parallelism is obtained via loop-level parallelism inside each node using compiler vased thread parallelisation techniques, such as OpenMP. HID/ExHID is applied to inter-node parallelization using message-passing interface (MPI), while two different methods are applied for intra-node parallelization using multi-threading(OpenMP). One is HID/ExHID and the other is multicolor-based reordering method. Both method are applied to distributed local data to yield the thread-level parallelism. We call the former strategy HID-HID since HID/ExHID is used for both intra-node and inter-node parallelization. And the latter is called HID-MC since HID/ExHID is used for intra-node parallelization and multicoloring is used for inter-node parallelization. Using OpenMP/MPI hybrid parallel programming model, we implemented finite element based simulations of linear elasticity problem solved by Krylov iterative solver with these hybrid parallel ILU(k) preconditioners. Developped codes are evaluated through strong scaling tests on multicore cluster called “T2K Open Supercomputer using up to 256 cores. Numerical experiments showed HID-HID generally leads to better convergence and less fill-ins. On the other hand, HID-MC could be more stable strategy than when increasing the number of threads per process.

The rest of this paper is organized as follows. Section 2 gives a overview of Hierarchical Interface Decompostion (HID) and parallel ILU preconditioning algrithm based on HID. And it descibes how fill-ins are introduced in the parallel ILU preconditioning algorithm by the extended HID. Section 3 describes tatget application based on finite element method and detailed implementations focusing on the thread level parallelization for two strategies, HID-HID and HID-MC. Section 4 contains the details on the test environment and numerical experiments followed by the conclusions section.

2 Hierarchical Interface Decompostion(HID) and Extended HID

By exploiting a static “hierarchical” decompostion of the graph, Hierarchical Interface Decompostion (HID) yeilds natural parallelism in the factorization and consider the global effect from external domain in parallel ILU/IC preconditioning process. However we cannot consider fill-ins from external domain in parallel ILU/IC preconditioning via HID. In order to consider fill-ins from external domains, we developed extended version of HID(ExHID). In this section we explain HID and our proposed method, ExHID. HID can be viewed as the methods from the angle of an ILU factorization combined with a form of nested dissection ordering in which cross points in the separators play a special role. The hierarchical decompostion starts with a partioning of the graph with one layer of overlap. Then “stages” or “levels” are defined from thie partioning, with each level (or stage) consisting of a set of vetex grpups (small connected subgraphs). Each vertex group of a given stage is a separator for the vertex groups of a lower stage. The incomplete factorization process proceeds level by level from lowerst to higherst. Due to the separation property of the vertex groups at different levels, this process can be carried out in a highly parallel manner. These vertex groups are called connectors in definition of HID. The concept of connectors of different levels and keys are introduced for the purpose of applying this idea to general graphs as follows:

– Connectors of level-1 (C^1) Are the sets of interior points. Each set of interior points is called a sub-domain.
– A connector of level-k (C^k) (k¿1) is adjacent to k sub-domains.
– Connectors in the same level never be adjacent to each other.
– Key(u) is the set of sub-domains (connectors of level-1, C^1) connected to vertex u.

Fig.1 (left) shows the example of the partition of a 9-point gird into 4 domains. In this case, there are 4 connectors of level-1 (C^1, sub-domain), 4 connectors of level-2 (C^2) and 1 connector of level-4 (C^4). Note that different connectors of the same level are not connected directly, but are separated by connectors of higher levels. These properties provide the block structure of the coeficient matrix A through reordering the unknowns by this decompostion. By reordring the unknowns according to their level numbers, from the lowest to highest, the block strucuture would be appeared as shown in Fig.1 (right). This block strucutre leads to natural parallelism in ILU/IC decomposition or forward/backward substitution processes. Fig.2 shows pseudo code of forward substitution in ILU(0) preconditioning. The most outer loop is for levels. At the end of each level global communication is performed according to hierarchical communication table. Hierachical communication table is communication table in which export/import nodes communicated between neighbor processes are arranged by level hierarchy. This communication performed at the end of each level transfers the update information calculated in the present level to the next level. Thus HID allows us to consider the global effect of external domains in parallel ILU preconditioning, which leads to more robust parallel preconditioning than block Jacobi-type localized preconditioners.

However global effect from external doamains which can be considered via HID is confined within the case of ILU(0)/IC(0). Fig.3 shows example of domain decomposition of two dimentional 9-point grid into two domains via HID (left) and how the local data are distributed on two processes. Though it depends on the numbering assined to node A and node B, these two nodes are in the distance which can affect each other when considering 2nd level of fill-ins. Suppose node A will affect on node B as a 2nd level of fill-in, we cannot take into account the effect from node A by the given distirubted local data sets since node A and node B are in the different distributed data sets. The first remedy is simply extending overlapped elements between domains. This allows us to consider the fill-in effect on node B from node A. But the fill-in effect from node A can only be calculated without any updates even which might have been occured on node A from other nodes related to A. Thus it fundamentally results the same as blcok jacobi-type preconditioning. Then thicker layer of separators is introduced in HID. Fig.4 illustrates the level-2 connector is extended from one layer to three layers. Then node A which was in level-1 connector in Fig.3, becomes in the level 2 connector. Since the nodes are reordered according to the level from lower to higher, ILU preconditioning process proceeds level by level from lower to higher. Node A in the lowe level is always calculated prior to node B. This allows us to calculate the fill-in effect on node B from node A with updates already calculated. Extension of layer of higher level connector allows us to consider fill-ins but also leads to load inbalance in general.

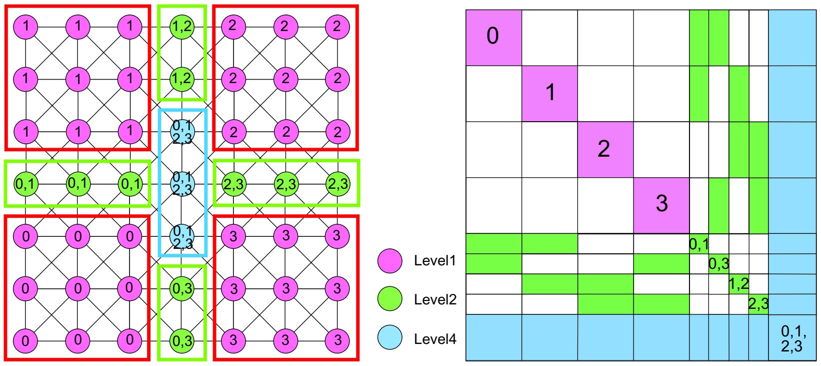

Fig. 1. Partitionoing of a 9 point grid into 4 subdomains by Hierarchical Interface Decompostion, resulted with four C^1 connectors(sub-domains), four C^2 connectors, one C^4 connectors. The numbers showed on the nodes are keys. The number of keys corresponds the level (left). Block structure is appeared in the coefficient matrix through HID reordering (right).

```
do lev= 1, LEVELtot
  do i= LEVindex(lev-1)+1, LEVindex(lev)
    SW1= WW(3*i-2,R); SW2= WW(3*i-1,R); SW3= WW(3*i  ,R)
    isL= INL(i-1)+1; ieL= INL(i)
    do j= isL, ieL
      k= IAL(j)
      X1= WW(3*k-2,R); X2= WW(3*k-1,R); X3= WW(3*k  ,R)
      SW1= SW1 - AL(9*j-8)*X1 - AL(9*j-7)*X2 - AL(9*j-6)*X3
      SW2= SW2 - AL(9*j-5)*X1 - AL(9*j-4)*X2 - AL(9*j-3)*X3
      SW3= SW3 - AL(9*j-2)*X1 - AL(9*j-1)*X2 - AL(9*j  )*X3
    enddo
    X1= SW1; X2= SW2; X3= SW3
    X2= X2 - ALU(9*i-5)*X1
    X3= X3 - ALU(9*i-2)*X1 - ALU(9*i-1)*X2
    X3= ALU(9*i  )*  X3
    X2= ALU(9*i-4)*( X2 - ALU(9*i-3)*X3 )
    X1= ALU(9*i-8)*( X1 - ALU(9*i-6)*X3 - ALU(9*i-7)*X2)
    WW(3*i-2,R)= X1; WW(3*i-1,R)= X2; WW(3*i  ,R)= X3
  enddo

  call SOLVER_SEND_RECV_3_LEV(lev,..):    Communications using
                                          Hierarchical Comm. Tables.
enddo
```

Fig. 2. Forward substitution process in preconditioning. Global communication is performed at the end of each level, which allows us to caluculate the next level using updated data from previous level. This makes us parallel ILU(0) more consistent by HID than Block Jacobi-type localized method.

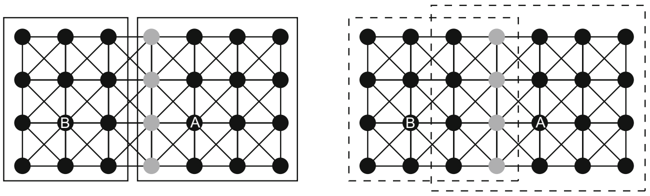

Fig. 3. Internal nodes assigned to two processes by domain decomposition via HID (left). There are two level-1 connectors (C^1, sub-domains) shown by Black nodes and one level-2 connector(C^2). The distributed local mesh is given with one overlapped layer (right).

3 Test Application and the Implementation

In this section test problems is discribed and the detailed implementations focusing on the intra-node parallelization are explained on each method employed in our two strategies, HID-HID and HID-MC.

3.1 Finite Element Based Simulations of Linearelasticity Problems

The test problem is finite element based simulations of three dimensional linearelasticity problem. Simple cube shaped analysis model is discritised by tri-linear hexahedral elements. Poisson's ratio and Young's modulus are given homogeneously for all elements and set to 0.25 and 1.0 respectively. The boundary

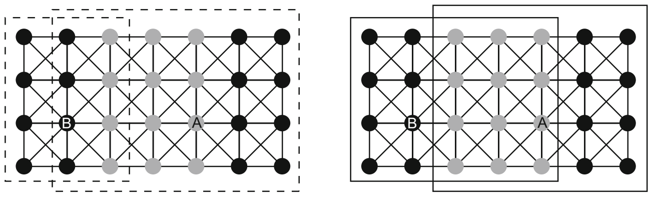

Fig. 4. The distributed local data sets when introducing thicker separator in HID (left). The level-2 connector (C^2, gray nodes) is extended to three layers. Thicker separator expand the range for global operations (right).

conditions are described in Fig.5. The Generalized Productive-type BiCG iterative solver with ILU(k) preconditioner is applied. Iterations are repeated until the norm $\|r\| / \|b\|$ is less than 10^{-8}. The code is based on the framework for parallel FEM procedures of GeoFEM[9], and GeoFEM's local data structure is applied. The local data strucutres in GeoFEM are node-based with overlapping elements[9]

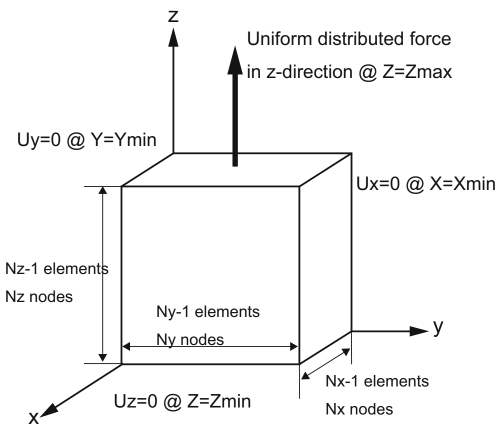

Fig. 5. Test model of simple cube geometry and the boundary conditions

3.2 HID/ExHID Ordering for Distributed Data

To yield the parallelism in the each distributed local data sets obtained by HID, we again apply HID/ExHID to the disributed data sets. We apply HID for ILU(0) and ExHID for ILU(k) to the disributed data sets on each MPI process and the resulted sub-domains (C^1) are assigned to threads and the adjacent C^k connectors to C^1s are also distributed among threads. Thus the number of sub-domains resulted by HID/ExHID corresponds to the number of threads in our implementation. And in our implementation of ExHID, extension of connector is applied in only level-2 connectors. No extensions in C^k connectors ($k > 2$).

Depending on the level of fill-ins to be considered we set the thickness. Fig.6 (right) illustrates our implementation of ExHID for the same example of 2D 9-point grid mesh. The dashed line shows the distribution of conncectors among threads. In Fig.6 (left) the decomposition by HID is illustlated for comparison. Without thicker separator node A and node B are in the same level in HID (left). Thus they are to be processed in parallel by different threads. If consider the fill-in effect on node A from node B, no updates on node B are avairable for node A. Moreover this trigger the data dependency problem between threads since one thread having node A is going to read the data on node B to calculate fill-ins on node A while another thread having node B is going to write the data on node B. By introducing thicker separator as in Fig.6 (right) put node B in level-2. Nodes in the higher level are processed after the nodes in the lower level so node B is processed after node A. This removes the data dependency and allows us to calculate the fill-in effect with update information.

However thicker separator is now applied for only level-2 connector, the same data dependency problem can happen between nodes in the higher level connectors than level 2, for example node C and node D in Fig.6 (right). Although they are in the distance which can affect when considering the 2nd level of fill-ins, they are in the same level. If these nodes are assigned to different threads, the same data dependency occurs between the threads. For avoiding such possible data dependencies in high level connectors, we ignore the fill-in effect from the node in the same level but on the different thread in our implementation.

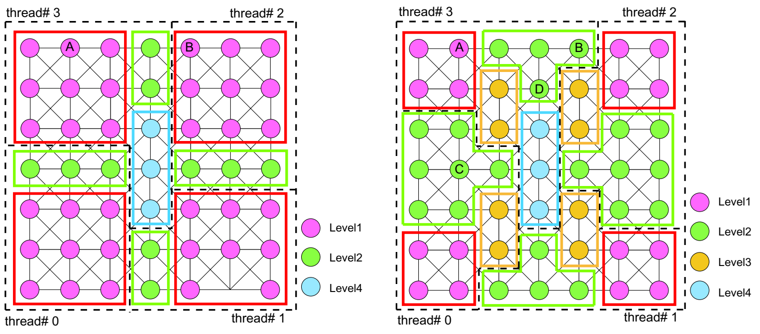

Fig. 6. The partitioning of a 9 point grid in to 4 subdomains by HID (left) and that by ExHID (right). Introducing thicker level-2 connector allows us to consider the fill-in effect on node A from node B and it also remove data dependency on it. Dashed line shows how connectors are distributed among threads.

3.3 Multicoloring Based Ordering for Distributed Data

As another parallelization method applied to distributed local data set is multicoloring, which is commonly used for parallelization of ILU factorizations. For taking into account the effect of fill-ins we apply the coloring rule which becomes strict according to level of fill-ins. For example, if we don't consider fill-ins at all, it is enough to color the nodes avoiding the adjacent nodes being in the same color. On the contrary, we apply the coloring rule so that every node are in different color from its neighbors' and "neighbors' of neighbors". Thus the number of colors increases in accordance with the level of fill-ins considered.

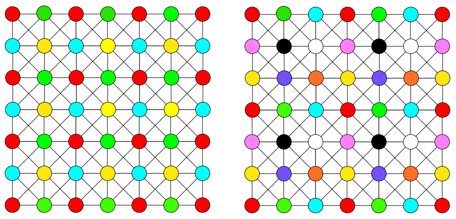

Fig. 7. Coloring rule for ILU(0) is simply to chose the color for each node avoiding the node in the same color with its adjacencies (right). For parallelization of ILU(1) each node has to be colored avoiding it in the same color with its adjacencies and the adjacencies of its adjacencies. The more colors are needed acording to the level of fill-ins to be considered.

3.4 Optimization for Memory Access

In the developed code, first touch data placement is considered. And appropriate command lines for NUMA control is applied, which is suppoted by Linux system, for efficient memory access to local memory. Minimizing memory access overhead is important for cc-NUMA architecture, such as T2K/Tokyo[10]. In order to reduce memory traffic in the system, it is important to keep the data close to the cores that runs with the data. On cc-NUMA architecture, this corresponds to making sure the pages of memory are allocated and owned by the core that works with the data contained in the page. The most common cc-NUMA page-placement algorithm is the first touch algorithm[11], in which the core first referencing a region of memory has the page holding that memory assigned to it. Very common technique in OpenMP program is to initialize data in parallel using the same loop schedule as it will be used lated in the computations.

4 Numerical Experiments

We tested two different types of hybrid parallel strategies HID-HID and HID-MC for solving the same problem discribed in section. To compare the performance for hybrid parallel method as iterative solver with ILU(k) preconditioing, we execute two numerical experiments. In both experiments we applied a strong scaling. The first test is run to compare the performances of these strategies using up to 16 nodes (256 cores) where the problem size is fixed at $3,090,903$ DOF (100^3 elements). The hybrid programming model applied is also fixed as 4x4 through this test. The second test is run to see their perfromances when the number of threads per process is incrased. The number of processes is fixed at eight and the problem size is fixed at $1,590,000$ DOF (80^3 elements).

4.1 Hardware Environment

Test environment is “T2K Open Super conputer (Todai Combined Cluster) (T2K/Tokyo), which was developed by Hitach under “T2K Open Supercomputer Alliance”[12]. T2K/Tokyo is an AMD Quad-core Opteron based combined cluster system with 952 nodes, 15,232 cores and 31 TB memory. Total peak performance is 140 TFLOPS. T2K/tokyo is an integrated system of four clusters. Number of nodes in each cluster is 512, 256, 128 and 56 respectively. Each node includes four “sockets” of AMD Quad-core Opteron processors(2.3GHz), as shown in Fig. Peack performance of each core is 9.2 GFLOPS and that of each node is 147.2 GFLOPS. Each node is connected via Myrinet-10G network. In the present work, up to 64 nodes of the system have been used. Because T2K/Tokyo is based on cache-coherent NUMA (cc-NUMA) architecture, careful design of software and data configuration is required for efficient memory access to local memory as stated in the previous section. We applied 4x4 hybrid programming model(four MPI processes x four OpemMP threads where one MPI process per one socket and four OpenMP threads per one MPI process) which is the most efficient case for this type of application on T2K.

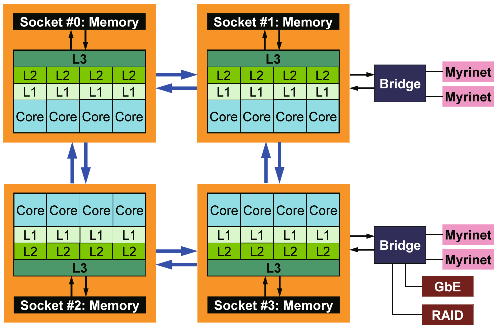

Fig. 8. The node specification of T2K/Tokyo. Each node includes four “sockets” of AMD Quad-core Opteron processors(2.3GHz). 16 cores per node.

4.2 Configuration of Hybrid Parallel Executions

We consider the fill-ins up to 2nd level of fill-ins. Thus ILU(0), ILU(1), and ILU(2) are applied to the iterative solver. In HID-HID strategy, we apply HID(3) for both ILU(1) and ILU(2). HID(3) is ExHID whose level-2 connector is extended to three layer of thickness. On the other hand, the number of colors required for parallelization of ILU(k) become larger the fill-in level k increases. In HID-MC strategy we set the number of colors for to the minimum number of colors required to yeild parallelism. The number of colors tested in the numerical experiments are eight for ILU(0), 27 for ILU(1), 64 for ILU(2).

4.3 Strong Scaling Test Up to 256 Cores

Fig.9 shows a comparison of elapsed time per iterations between HID-HID and HID-MC as increasing the number of nodes under 4x4 hybrid programming model where the problem size is fixed at $3,000,000$ DOF. For parallelization of ILU(k), we apply HID(3) while the number of colors in HID-MC is set to the minimum. Up to 256 cores(16 nodes), the similer scalabilities are observed for both strategy. Fig.10 and Fig.11 shows another comparisons on convergence and memory requirement between HID-HID and HID-MC. Fig.10 shows the iterations required for convergence and Fig.11 shows the number of fill-ins occured for ILU(1) and ILU(2). HID-HID leads to smaller values in both iterations and memory than HID-MC. Finally Fig.12 show the comparison on total solver time (elapsed time for total iterations) between HID-HID and HID-MC. Due to the larger iterations and larger number of fill-ins which is directly related to the computational the total solver time becomes larger by HID-MC than HID-HID larger iterations.

4.4 Strong Scaling Test with Different Hybrid Programming Models

Fig.13 shows solver time (elapsed time for total iterations) as the number of threads per process incrases. The number of threads is increased from one to 16 while process is fixed at eight. The problem size is here $1,590,000$ DOF. Fig.13 again shows it cost longer time by HID-MC than HID-HID. Fig.14 shows the iterations required for convergence as the number of threads per process increases.

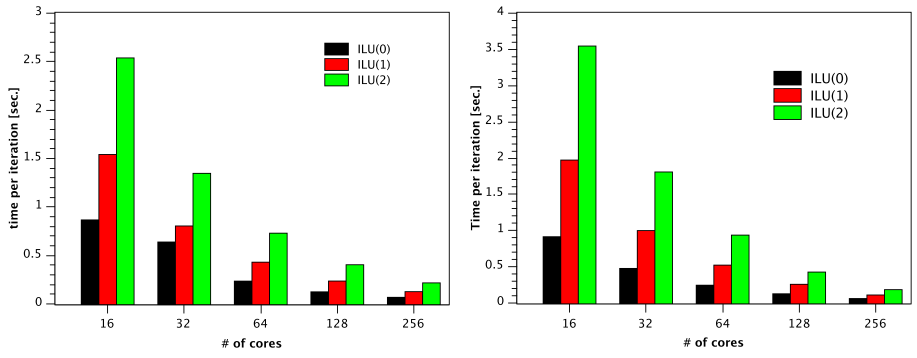

Fig. 9. Time per iteration with increase of the number of nodes under 4x4 hybrid parallele programming on the 100^3 elements problem. GPBiCG preconditioner with ILU(0), ILU(1), and ILU(2) preconditioner are applied. Intra-node parallelization for ILU(1) and ILU(2), HID(3) (ExHID adopted with thickness three) are used in HID-HID. On the other hand, the minimum number of colors, 27 colors for ILU(1) and 64 colors for ILU(2), is set in HID-MC. The result by HID-HID is shown in the left and HID-MC in the right.

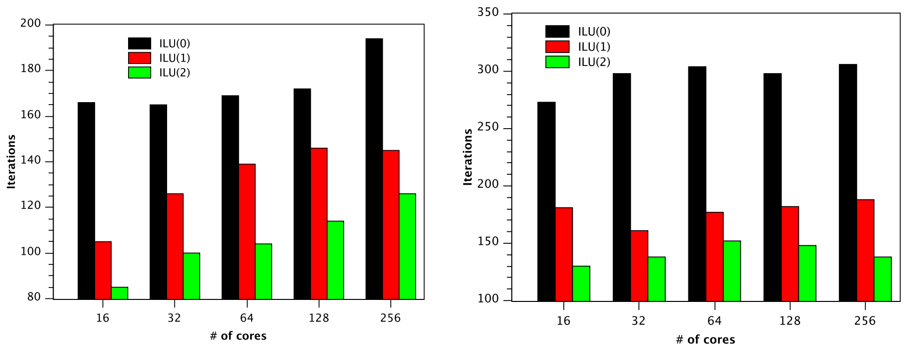

Fig. 10. Iteration with increase of the number of nodes under 4x4 hybrid parallele programming. GPBiCG preconditioner with ILU(0), ILU(1), and ILU(2) preconditioner are applied. Intra-node parallelization for ILU(1) and ILU(2), HID(3) (ExHID adopted with thickness three) are used in HID-HID. On the other hand, the minimum number of colors, 27 colors for ILU(1) and 64 colors for ILU(2), is set in HID-MC. The result by HID-HID is shown in the left and HID-MC in the right.

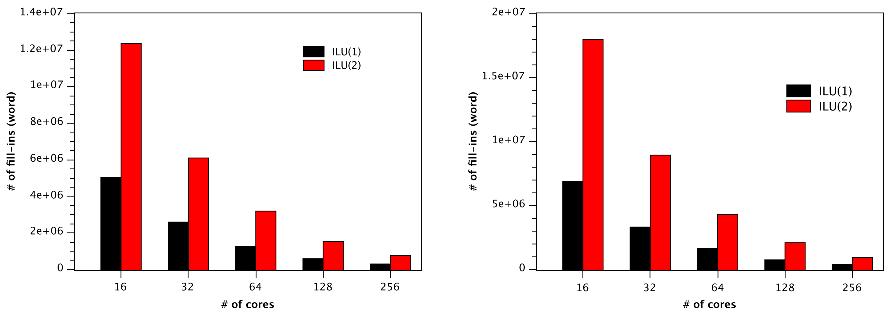

Fig. 11. The number of Fill-ins with increase of the number of nodes under 4x4 hybrid parallele programming which is resulted in ILU(1)/ILU(2) for each strategy. The result by HID-HID is shown in the left and HID-MC in the right.

The iterations are also larger by HID-MC than HID-HID. But it is observed that the more iterations are required for ILU(1) and ILU(2) case as increasing the number of therads per processe in HID-HID, while those stay almost the same for the number of threads in HID-MC. This is because our ILU(k) implementation based on ExHID can only avoid data dependency between C^1 connectors. In our implementation, we ignore the fill-in effect when the fill-in node has data dependency (i.e. the fill-in node is assigned to different thread and is in the same level) from higher level connectors, as discribed in section3.2. Such nodes to be ignored due to data dependency exist more and more in the higher level when the number of C^1 connectors increases (i.e. the number of threads increases).

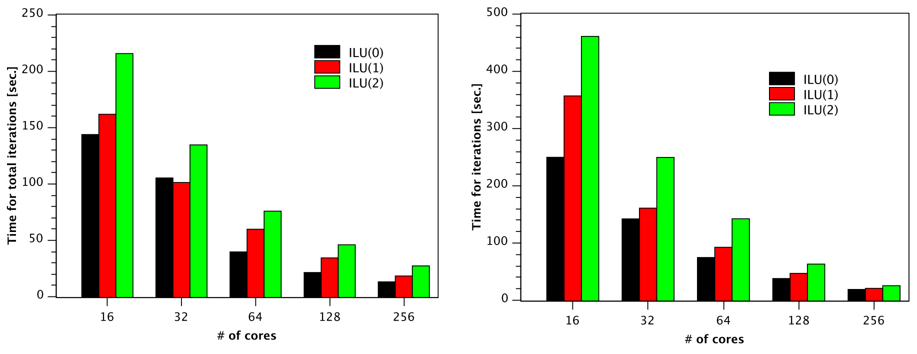

Fig. 12. Solver time (elapsed for total iterations) with increase of the number of nodes under 4x4 hybrid parallele programming. GPBiCG preconditioner with ILU(0), ILU(1), and ILU(2) preconditioner are applied. Intra-node parallelization for ILU(1) and ILU(2), HID(3) (ExHID adopted with thickness three) are used in HID-HID. On the other hand, the minimum number of colors, 27 colors for ILU(1) and 64 colors for ILU(2), is set in HID-MC. The result by HID-HID is shown in the left and HID-MC in the right.

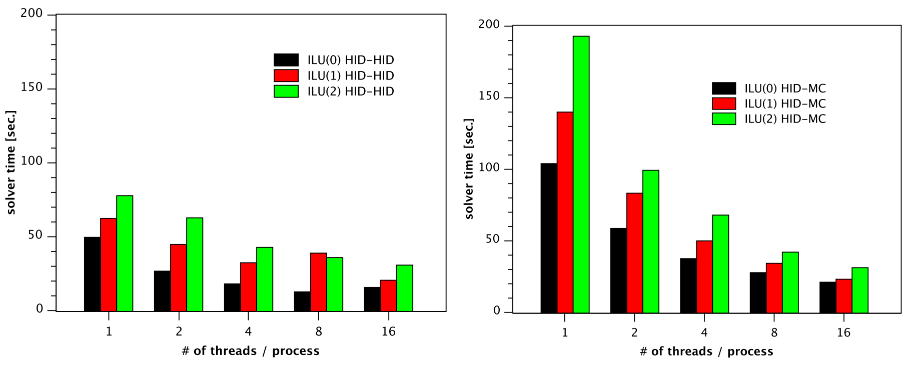

Fig. 13. Solver time (elapsed for total iterations) with increase of the number of threads per processes by GPBiCG with ILU(0), ILU(1), and ILU(2) for problem of 80^3 elements. The number MPI process is fiexed at eitht. 80^3 elements. HID-HID case is shown in left and HID-MC in right.

4.5 Number of Colors in HID-MC

For HID-MC we set the minimum number of colors needed for parallelization in previous execution. Now we increase the number of colors and compare the convergence and computation time between HID-HID and HID-MC. Fig.15 shows iterations and solver time for ILU(0)+GPBiCG executed by Hybrid 4x4 case using 16 nodes(256 cores) on small test model (80^3 elements, $1,594,323$ DOF).

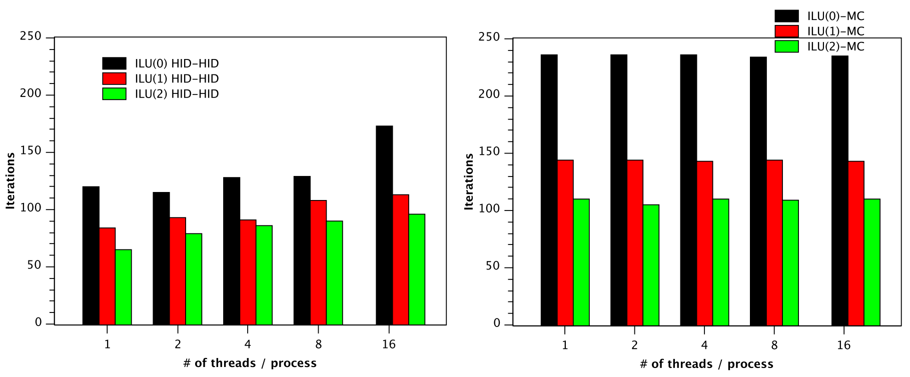

Fig. 14. Iterations with increase of the number of threads per processes by GPBiCG with ILU(0), ILU(1), and ILU(2) for problem of 80^3 elements. The number MPI process is fiexed at eitht. 80^3 elements. HID-HID case is shown in left and HID-MC in right.

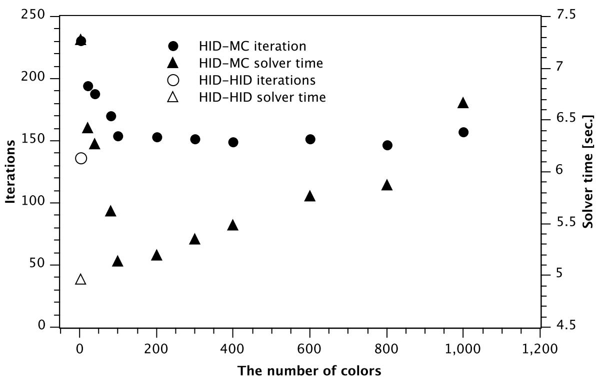

Fig. 15. 80^3elements ILU(0)+GPBiCG, Intranode parallelization for ILU(0) HID-HID (no thicker separator), HID-MC (color number 8(minimum))

As Fig.15 displays, the iterations become shorter, while the solving time become shorter once and gradually become larger again in accordance with increase of the number of colors. This is simply because the larger number of colors leads the more frequent syncronization among threads. Thus the performance of HID-MC is directly related to the number of colors and if we can find the optimal number of colors (in this case 100 is the optimal), HID-MC can archive the close performance to HID-HID.

5 Concluding Remarks

ILU(k) preconditioner is widely used powerful preconditioner in many finite element applications and it is important to establish the hybrid parallel scheme for ILU(k) preconditioner. HID and ExHID is a robust and effective parallelization method of ILU(k) and we developed hybrid parallel scheme for ILU(k) preconditioner based on HID and ExHID. For intra-node parallelization we can have several variations of methods. In order to find more efficient strategy for hybrid parallel ILU(k), we implemented using two differetn strategies, HID-HID and HID-MC. By applying OpenMP/MPI hybrid programming model, our two implementation are evaluated on multi-core cluster using up to 256 cores. HID-HID strategy leads better convergence and fewer fill-ins than HID-MC generally. However towarding many core environment more than 100, HID-MC can be more stable strategy than HID-HID. Multicoloring brings us another task how to find optimal number of colors but the flexibiliy of multicoloring which is easily applicable to the case of the large number of threads becomes advantage to HID-HID. Depending the test environment, tactical selection of hybrid strategy is important for ILU(k) preconditioner on multi/many- core clusters.

References

1. Jones, M.T., Plassmann, P.E.: Scalable iterative solution of sparse linear systems. Parallel Computing 20(5), 753–773 (1994)
2. Saad, Y., Sosonkina, M.: Distributed schur complement techniques for general sparse linear systems. SIAM Journal on Scientific Computing 21(4), 1337–1356 (2000)
3. Parallel iterative solvers of geofem with selective blocking preconditioning for nonlinear contact problems on the earth simulator (2003)
4. The Impact of Parallel Programming Models on the Linear Algebra Performance for Finite Element Simulations (2007)
5. Washio, T., Hisada, T., Watanabe, H., Tezduyar, T.E.: A robust preconditioner for fluid-structure interaction problems. Computer Methods in Applied Mechanics and Engineering 194(39-41), 4027–4047 (2005)
6. Nakajima, K.: Parallel preconditioning methods with selective fill-ins and selective overlapping for ill-conditioned problems in finite-element methods. In: Shi, Y., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2007, Part III. LNCS, vol. 4489, pp. 1085–1092. Springer, Heidelberg (2007)
7. Saad, Y., Sosonkina, M.: Enhanced parallel multicolor preconditioning techniques for linear systems. UMSI Research Report/University of Minnesota (Minneapolis, Mn). Supercomputer Institute, vol. 99, p. 4 (1999)
8. Henon, P., Saad, Y., et al.: A parallel multistage ilu factorization based on a hierarchical graph decomposition. SIAM Journal on Scientific Computing 28(6), 2266 (2006)
9. `http://geofem.tokyo.rist.or.jp/`
10. Nakajima, K.: Flat mpi vs. hybrid: Evaluation of parallel programming models for preconditioned iterative solvers on "t2k open supercomputer". In: International Conference on Parallel Processing Workshops, ICPPW 2009, pp. 73–80 (2009)
11. Mattson, T., Sanders, B., Massingill, B.: Patterns for parallel programming. Addison-Wesley Professional (2004)
12. `http://www.opensupercomputer.org/`