Parallel Multigrid Solvers using OpenMP/MPI Hybrid Programming Models on Multi-Core/Multi-Socket Clusters

Kengo Nakajima[1]

[1] Information Technology Center, The University of Tokyo, 2-11-16 Yayoi, Bunko-ku, Tokyo 113-8658, Japan
nakajima@cc.u-tokyo.ac.jp

Abstract. OpenMP/MPI hybrid parallel programming models were implemented to 3D finite-volume based simulation code for groundwater flow problems through heterogeneous porous media using parallel iterative solvers with multigrid preconditioning. Performance and robustness of the developed code has been evaluated on the "T2K Open Supercomputer (Tokyo)" and "Cray-XT4" using up to 1,024 cores through both of weak and strong scaling computations. OpenMP/MPI hybrid parallel programming model demonstrated better performance and robustness than flat MPI with large number of cores for ill-conditioned problems with appropriate command lines for NUMA control, first touch data placement and reordering of the mesh data for contiguous "sequential" access to memory.

Keywords: Multigrid, OpenMP/MPI Hybrid, Preconditioning

1 Introduction

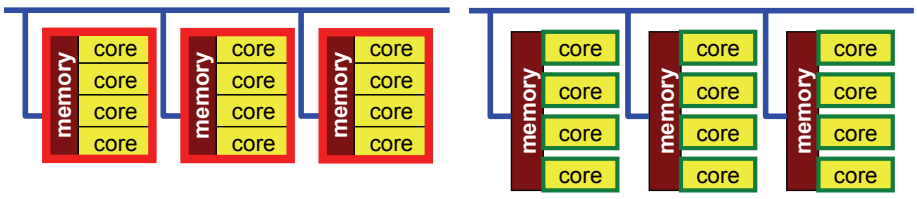
In order to achieve minimal parallelization overheads on SMP (symmetric multiprocessors) and multi-core clusters, a multi-level *hybrid* parallel programming model is often employed. In this method, coarse-grained parallelism is achieved through domain decomposition by message passing among nodes, while fine-grained parallelism is obtained via loop-level parallelism inside each node using compiler-based thread parallelization techniques, such as OpenMP (Fig.1 (a)). Another often used programming model is the single-level *flat MPI* model, in which separate single-threaded MPI processes are executed on each core (Fig.1 (b)). It is well-known that the efficiency of each model depends on hardware performance (CPU speed, communication bandwidth/latency, memory bandwidth/latency, and their balance), application features, and problem size [1].

In the previous work [1], author applied OpenMP/MPI hybrid parallel programming models to finite-element based simulations of linear elasticity problems. The developed code has been tested on the "T2K Open Supercomputer" [2] using up to 512 cores. Performance of OpenMP/MPI hybrid parallel programming model is competitive with that of *flat MPI* using appropriate command lines for NUMA control.

Furthermore, reordering of the mesh data for contiguous access to memory with first touch data placement provides excellent improvement on performance of OpenMP/MPI hybrid parallel programming models, especially if the problem size for each core is relatively small. Generally speaking, OpenMP/MPI hybrid parallel programming model provides excellent performance for strong scaling cases where problems are *less memory-bound*.

In the present work, OpenMP/MPI hybrid parallel programming models were implemented to 3D finite-volume based simulation code for groundwater flow problems through heterogeneous porous media using parallel iterative solvers with multigrid preconditioning, which was originally developed in [3]. Multigrid is a scalable method and expected to be a promising approach for large-scale computations, but there are no detailed research works where multigrid methods are evaluated on multi-core/multi-socket clusters using OpenMP/MPI hybrid parallel programming models. In this work, developed code has been evaluated on the "T2K Open Super Computer (Todai Combined Cluster) (T2K/Tokyo)" at the University of Tokyo, and "Cray-XT4" at National Energy Research Scientific Computing Center (NERSC) of Lawrence Berkeley National Laboratory [4] using up to 1,024 cores, and performance of *flat MPI* and three kinds of OpenMP/MPI hybrid parallel programming models are evaluated.

The rest of this paper is organized as follows: In section 2, overview of the target hardware ("T2K/Tokyo", "Cray XT4") is provided. In section 3, we outline the details of the present application, and describe the linear solvers and reordering/optimization procedures. In section 4, preliminary results of the computations for both of weak and strong scaling tests are described, while some final remarks are offered in section 5.



(a) Hybrid (b) Flat MPI

Fig.1. Parallel Programming Models

2 Hardware Environment

"T2K Open Super Computer (Todai Combined Cluster) (T2K/Tokyo)" at the University of Tokyo [2] was developed by Hitachi under "T2K Open Supercomputer Alliance" [5]. T2K/Tokyo is an AMD Quad-core Opteron-based combined cluster system with 952 nodes, 15,232 cores and 31TB 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.2.

Peak 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, 64

nodes of the system have been evaluated. Because T2K/Tokyo is based on *cache-coherent* NUMA (cc-NUMA) architecture, careful design of software and data configuration is required for efficient access to local memory.

"Cray XT4 (Franklin)" system at NERSC/LBNL [4] is a large-scale cluster system. A single node of Cray-XT4 corresponds to a single "socket" of AMD Quad-core Opteron processor (2.3 GHz) in Fig.2. Entire system consists of 9,572 nodes, 38,288 cores and 77TB memory. Total peak performance is 352 TFLOPS. Network topology of T2K/Tokyo is multi-stage cross-bar, while that of Cray-XT4 is 3D truss.

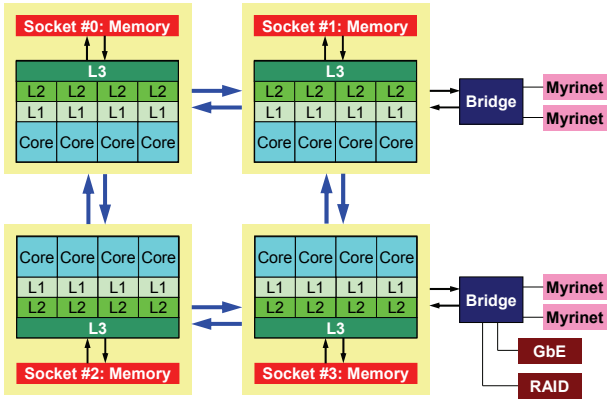


Fig. 2. Overview of a "node" of T2K/Tokyo, each node consists of four sockets of AMD Quad-core Opteron processors (2.3GHz)

3 Implementation and Optimization of Target Application

3.1 Finite-Volume Application

In the present work, groundwater flow problems through heterogeneous porous media (Fig.3) are solved using a parallel finite-volume method (FVM). Problem is described by the following Poisson equation and boundary condition:

$$\nabla \cdot (\lambda(x,y,z)\nabla\phi) = q, \phi = 0 \, at \, z = z_{\max} \quad (1)$$

where ϕ denotes potential of water-head, and $\lambda(x,y,z)$ describes water conductivity. q is value of volumetric flux of each cell, and is set to a uniform value (=1.0) in this work. A heterogeneous distribution of water conductivity in each cell is calculated by a sequential Gauss algorithm, which is widely used in the area of geostatistics [6]. The minimum and maximum values of water conductivity are 10^{-5} and 10^{5}, respectively, with an average value of 1.0. This configuration provides ill-conditioned coefficient matrices whose condition number is approximately 10^{10}. Each cell is a cube, and distribution of cells is structured like finite-difference-type voxels . In this work, entire model is consists of clusters of small models with 128^3 cells. In each small model, distribution pattern is same therefore same pattern of heterogeneity

appears periodically. The code is parallelized by domain decomposition using MPI for communications between partitioned domains [3].

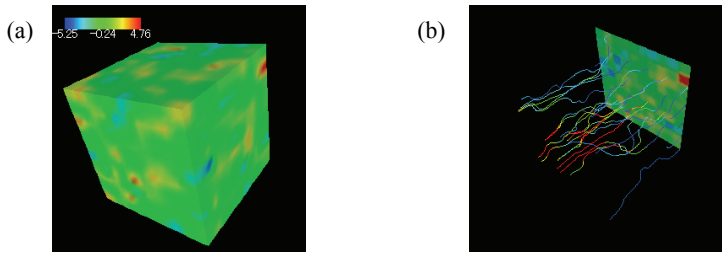


Fig.3. Example of groundwater flow through heterogeneous porous media, (a) distribution of water conductivity, (b) streamlines

3.2 Iterative Method with Multigrid Preconditioning

The Conjugate Gradient (CG) solver with multigrid preconditioner (MGCG) [3] was applied for solving Poisson's equations with symmetric positive definite (SPD) coefficient matrices. Iterations are repeated until the norm $|r|/|b|$ is less than 10^{-12}. Multigrid is an example of scalable linear solver and widely used for large-scale scientific applications. Relaxation methods such as Gauss-Seidel can efficiently damp high-frequency error, but low-frequency error is left. The multigrid idea is to recognize that this low-frequency error can be accurately and efficiently solved for on a coarser grid [7]. In this work, very simple geometric multigrid with V-cycle, where 8 children form 1 parent cell in isotropic manner for structured finite-difference-type voxels, as shown in Fig.4, has been applied. *Level* of the finest grid is set to 1 and the *level* is numbered from the finest to the coarsest grid. In this work, multigrid operations at each level are done in parallel manner, but the operations at the coarsest levels are executed on a single core by gathering information of entire processes.

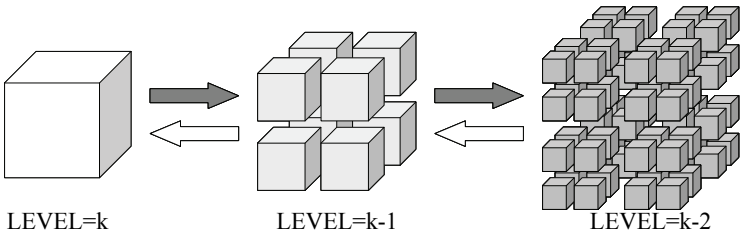


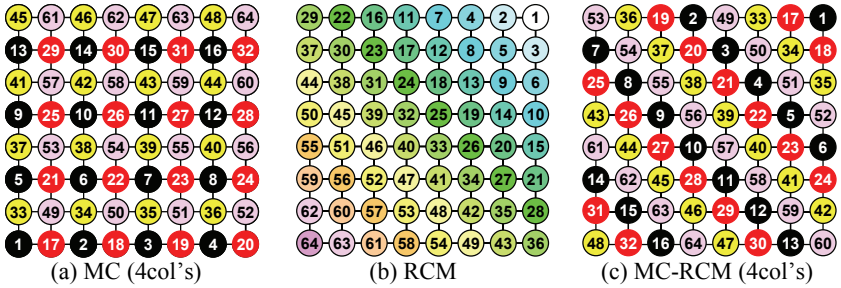
Fig.4. Procedure of Geometric Multigrid (8 children = 1 parent)

In multigrid procedure, equations at each level are "relaxed" using smoothing operators, such as Gauss-Seidel iterative solvers. Many types of smoothing operators have been proposed and used [7]. Among those, ILU(0)/IC(0) (Incompolete LU/Cholesky factorization without fill-in's) are widely used. These smoothing operators demonstrate excellent robustness for ill-conditioned problems [3,7,8]. In this study, IC(0) is adopted as a smoothing operator. IC(0) process includes global operations and it is difficult to be parallelized. Block-Jacobi-type localized procedure is possible for distributed parallel operations, but this approach tends to be unstable

for ill-conditioned problems. In order to stabilize the localized IC(0) smoothing, additive Schwarz domain decomposition (ASDD) method for overlapped regions [9] has been introduced.

3.3 Procedures for Reordering

The 3D code is parallelized by domain decomposition using MPI for communications between partitioned domains. In the OpenMP/MPI hybrid parallel programming model, multithreading by OpenMP is applied to each partitioned domain. The reordering of elements in each domain allows the construction of local operations without global dependency, in order to achieve optimum parallel performance of IC operations in multigrid processes.



(a) MC (4col's) (b) RCM (c) MC-RCM (4col's)

Fig.5. Various Methods for Coloring

Reverse Cuthill-McKee (RCM) reordering facilitates a faster convergence of iterative solvers with ILU/IC preconditioners than traditional multicolor (MC) reordering, especially for ill-conditioned problems, but leads to irregular numbers of vertices in each level set. The solution to this trade-off is RCM with cyclic-multicoloring (CM-RCM) [11]. In this method, further renumbering in a cyclic manner is applied to vertices that are reordered by RCM, as shown in Fig.5 (c). In CM-RCM, the number of colors should be large enough to ensure that vertices of the same color are independent.

3.4 Procedures for Optimization

In the current work, following three types of optimization procedures have been applied to OpenMP/MPI hybrid parallel programming models:

- Appropriate command lines for NUMA control
- First touch data placement
- Reordering for contiguous "sequential" access to memory,

Same command lines for NUMA control as were used in [1] have been applied in the current work. Detailed information for optimum command lines can be found in [1].

3.4.1 First Touch Data Placement

Minimizing memory access overhead is important for cc-NUMA architecture, such as T2K/Tokyo. In order to reduce memory traffic in the system, it is important to keep the data close to the cores that works with the data. On cc-NUMA architecture, this corresponds to making sure the pages of memory are allocated and owned by the cores that works with the data contained in the page. The most common cc-NUMA page-placement algorithm is the *first touch* algorithm [10], in which the core first referencing a region of memory has the page holding that memory assigned to it. Very common technique in OpenMP programs is to initialize data in parallel using the same loop schedule as will be used later in the computations, as shown in Fig.6.

```
  do lev= 1, LEVELtot
    do ic= 1, COLORtot(lev)
!$omp parallel do private(ip,i,j,isL,ieL,isU,ieU)
      do ip= 1, PEsmpTOT
      do i = STACKmc(ip,ic-1,lev)+1, STACKmc(ip,ic,lev)
        RHS(i)= 0.d0; X(i)= 0.d0; D(i)= 0.d0

        isL= indexL(i-1)+1
        ieL= indexL(i)
        do j= isL, ieL
          itemL(j)= 0; AL(j)= 0.d0
        enddo

        isU= indexU(i-1)+1
        ieU= indexU(i)
        do j= isU, ieU
          itemU(j)= 0; AU(j)= 0.d0
        enddo
      enddo
      enddo
!$omp omp end parallel do
    enddo
  enddo
```

Fig.6. Example of initialization of arrays for *first touch* data placement, where initialization process has been done

3.4.2 Reordering of Data for Contiguous "Sequential" Memory Access

In CM-RCM reordering, initial vector is re-numbered according to color ID, as shown in Fig.5. Elements in each color are distributed to each thread so that load for each thread is balanced.

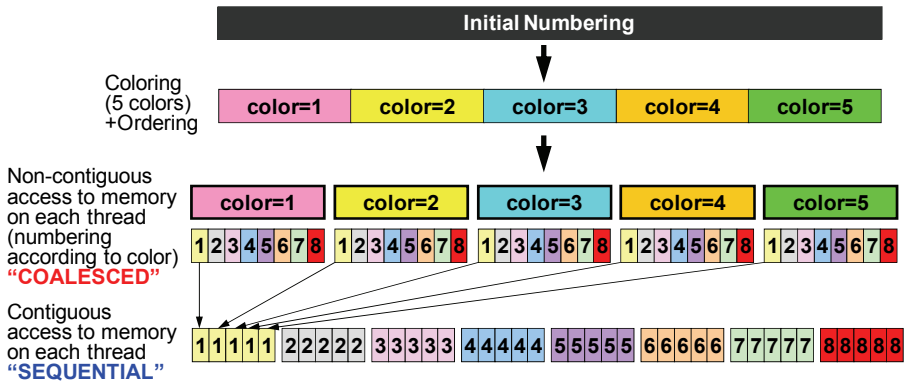


Fig.7. Data distribution on each thread after further reordering for contiguous "sequential" memory access, number of color: 5, number of thread: 8

Pages of memory are allocated to local memory of each socket through the *first touch* procedure for initialization described in Fig.6. But the problem is that the size of each page is small and addresses of pages in each thread are not contiguous, as shown in Fig.7. This provides inefficient performance of access to memory. In order to provide contiguous address of local pages, further reordering has been applied, as shown in Fig.7.

Thus, each thread can access pages of memory in contiguous manner. This pattern of memory access is called "sequential" and suitable for cc-NUMA architectures with multi-sockets and multi-cores, while original pattern of memory access in CM-RCM is "coalesced" which is rather favorable in GPU computing.

4 Results

4.1 Effect of Coloring and Optimization

Performance of the developed code has been evaluated using between 16 and 1,024 cores of the T2K/Tokyo and Cray-XT4. IC(0) smoothing is applied twice at each level with a single cycle of ASDD at each smoothing operation. A single V-cycle operation is applied at a preconditioning process of each CG iteration.

Following three types of OpenMP/MPI hybrid parallel programming models are applied as follows, and results are compared with those of *flat MPI*:

- **Hybrid 4×4 (HB 4×4)**: Four OpenMP threads for each of four sockets in Fig.2, four MPI processes in each node, both of T2K/Tokyo and " and Cray-XT4
- **Hybrid 8×2 (HB 8×2)**: Eight OpenMP threads for two pairs of sockets in Fig.2, two MPI processes in each node, only for T2K/Tokyo
- **Hybrid 16×1 (HB 16×1)**: Sixteen OpenMP threads for a single node in Fig.2, one MPI process in each node, only for T2K/Tokyo

Because each node of Cray-XT4 has a single socket with four cores, only HB 4×4 has been applied to Cray-XT4 as OpenMP/MPI hybrid cases.

First of all, effect of reordering and optimization for OpenMP/MPI hybrid cases described in the previous chapter has been evaluated using 4 nodes (64 cores) of T2K/Tokyo for *flat MPI* and OpenMP/MPI hybrid parallel programming models. Number of finite-volume cells per each core is 262,144 ($=64^3$), therefore total problem size is 16,777,216. Figure 8 provides relationship between number of iterations for convergence of MGCG solvers with CM-RCM reordering for each parallel programming model and number of colors for CM-RCM reordering. Generally speaking, number of iterations for convergence of iterative solvers with IC/ILU-type preconditioners decreases, as number of colors increases, according to the theory of "incompatible nodes" described in [11]. Convergence of the problem in this work generally follows this theory, as shown in Fig.8.

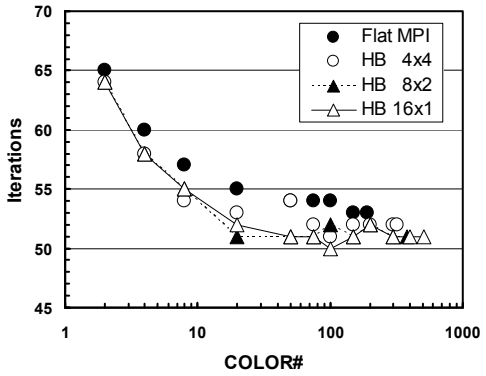


Fig.8. Performance of MGCG solver with CM-RCM reordering on T2K/Tokyo, 16,777,216 cells, 64 cores, Number of Iterations for Convergence

Each of Fig. 9 (a) and (b) provides the relationship between performance (computation time for linear solvers) and number of colors for each parallel programming model. The procedures for optimization described in the previous chapter for OpenMP/MPI hybrid cases have been already applied. Although number of iterations decreases according to increasing of number of colors, as shown in Fig.8, CM-RCM with only 2 colors (CM-RCM(2)) provides the shortest elapsed computation time of MGCG solvers for each parallel programming model, as shown in Fig.9 (a). In this type of geometry with structured finite-difference-type voxels, 2 colors are enough to ensure that vertices of the same color are independent in CM-RCM procedure. Figure 9 (b) shows computation time for MGCG solvers per iteration, and CM-RCM(2) provides the best computational performance (FLOPS rate). This is because that cache is more efficiently utilized if the number of colors is smaller in CM-RCM ordering for structured finite-difference-type voxels used in the current work. Each of Fig. 10 (a) and (b) shows an example of numbering of cells for 2D structured finite-difference-type voxels with 64 cells using (a) CM-RCM(2) (#1-#32 cells belong to the 1st color, while #33-#64 cells are in the 2nd color) and (b) RCM (with 15 colors), respectively. If forward/backward substitutions (FBS) during ILU operations are considered for cells of #29, #30 and #31 in CM-RCM(2) (Fig.10 (a)), numbering of off-diagonal variables (#59~#64) is contiguous, and diagonal and off-diagonal variables are on separate cache lines. On the contrast, corresponding diagonal and off-diagonal variables could be on a same cache line in RCM with 15 colors (Fig.10 (b)). In this case, the cache line is written back to memory, after one of the diagonal variables is updated in FBS process.

Figure 11 provides computation time of MGCG solver with CM-RCM (2) before and after optimization on T2K/Tokyo. Effect of optimization described in 3. (optimum command lines for NUMA control, first-touch data placement (Fig.6) and reordering of data for contiguous "sequential" memory access (Fig.7)) is significant, especially for HB 8×2 and HB 16×1. In HB 4×4, effect of NUMA control is significant, but effect of first-touch and "sequential" memory access is small, because all data for each process are guaranteed to be on local memory of each socket. *Flat MPI* and optimized OpenMP/MPI Hybrid cases are generally competitive from the viewpoint of computation time.

In *flat MPI* cases, time for reordering and set-up of matrices is approximately 1.50 sec. and 6.00 sec., respectively, while computation time for MGCG solvers is about 20 sec. In OpenMP/MPI hybrid cases of the current work, processes for reordering and set-up of matrices are not parallelized yet.

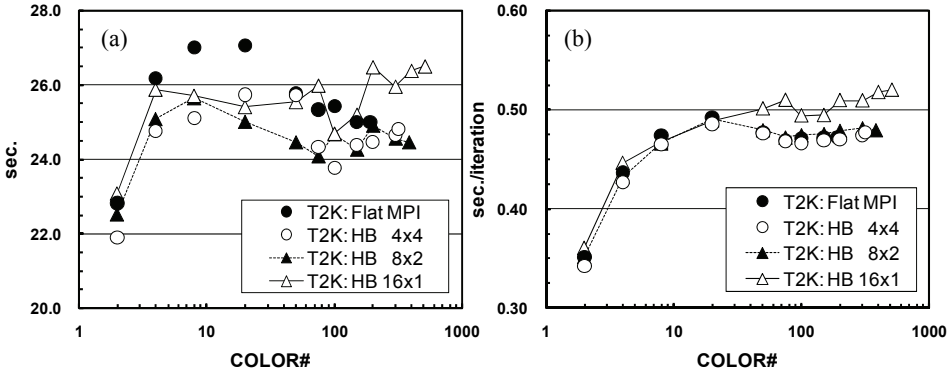


Fig.9. Performance of MGCG solver with CM-RCM reordering on T2K/Tokyo, 16,777,216 cells, 64 cores (Optimized solvers using first-touch data placement (Fig.6) and reordering of data for contiguous "sequential" memory access (Fig.7)), (a) Elapsed computation time for MGCG solvers, (b) Computation time for MGCG solvers for each iteration

(a)

45	10	39	5	35	2	33	1
17	46	11	40	6	36	3	34
53	18	47	12	41	7	37	4
24	54	19	48	13	42	8	38
59	25	55	20	49	14	43	9
29	60	26	56	21	50	15	44
63	**30**	61	27	57	22	51	16
32	64	**31**	62	28	58	23	52

(b)

29	22	16	11	7	4	2	1
37	30	23	17	12	8	5	3
44	38	31	24	18	13	9	6
50	45	39	32	25	19	14	10
55	51	46	40	33	26	20	15
59	56	52	47	41	34	27	21
62	**60**	57	53	48	42	35	28
64	63	**61**	58	54	49	43	36

Fig.10. Examples of numbering of cells for 2D structured finite-difference-type voxels using (a) CM-RCM (2) (#1-#32: 1st color, #33-#64: 2nd color), (b) RCM (with 15 colors)

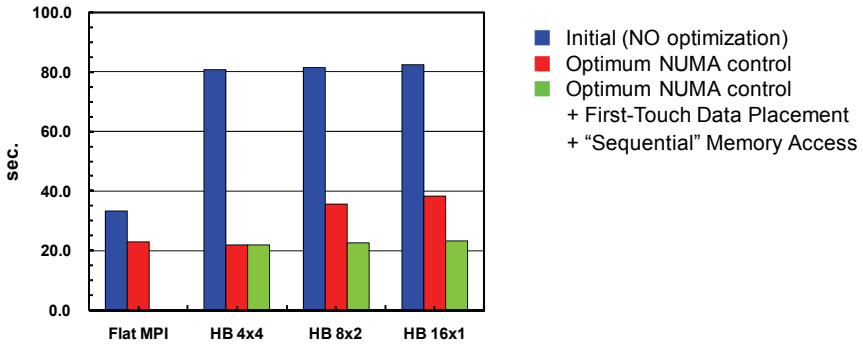


Fig.11. Performance of MGCG solver with CM-RCM(2) on T2K/Tokyo, 16,777,216 cells, 64 cores (Initial version of solvers, Solvers with optimum command lines for NUMA control, Solvers with additional optimization by first-touch data placement and reordering of data for contiguous "sequential" memory access), Computation time for MGCG solvers

4.2 Weak Scaling

Performance of weak scaling has been evaluated using between 16 and 1,024 cores of the T2K/Tokyo and Cray-XT4. Number of finite-volume cells per each core is 262,144 (=64^3), therefore maximum total problem size is 268,435,456. Figure 12 (a) shows computation time of MGCG solver until convergence, and Fig.12 (b) shows number of iterations for convergence. Number of iterations for convergence of *flat MPI* increases, as number of core is more than 256. On the contrast, number of iterations of OpenMP/MPI hybrid cases stays almost constant between 16 and 1,024 cores for the ill-conditioned problems with condition number of 10^{10}. This feature is more significant, as thread number per process increases. MGCG solvers with OpenMP/MPI hybrid parallel programming model provide excellent scalability even in this type of ill-conditioned problems. Generally speaking, robustness of localized IC(0) preconditioning is getting worse, as number of domains increases in ill-conditioned problems. OpenMP/MPI hybrid is generally more robust than *flat MPI* because number of cells at domain boundaries is relatively fewer.

Performance of Cray-XT4 is generally larger than that of T2K/Tokyo by 40%~50%. Memory performance of Cray-XT4 is about 25% larger than that of T2K/Tokyo according to STREAM benchmarks [12]. Furthermore, cache is utilized more efficiently on Cray-XT4 in multigrid operations especially for coarser level of cells, because no cache coherency is considered on Cray-XT4.

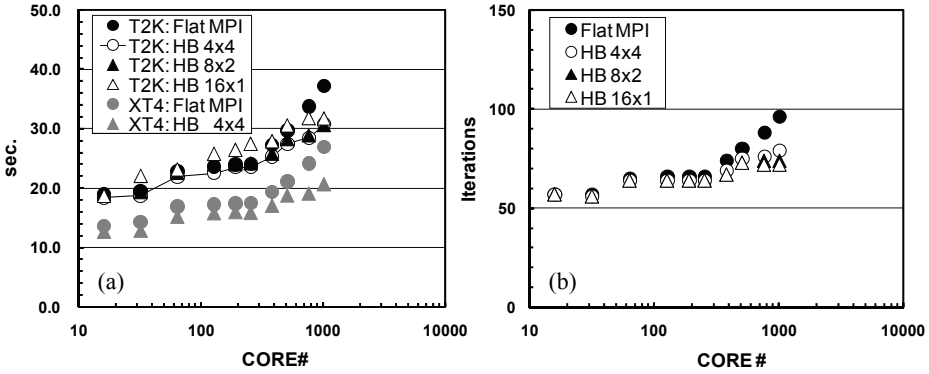


Fig.12. Performance of MGCG solver with CM-RCM(2) on T2K/Tokyo and Cray-XT4 using up to 1,024 cores, Weak Scaling: 262,144 cells/core, Max. Total Problem Size: 268,435,456 (a) Computation time for MGCG solvers, (b) Number of iterations for convergence

4.3 Strong Scaling

Performance of strong scaling has been evaluated for fixed size of problem with 33,554,432 cells (=512×256×256) using between 16 and 1,024 cores of T2K/Tokyo and Cray-XT4. Figure 13(a) shows relationship between number of cores and number of iterations until convergence for each parallel programming model. Number of iterations for *flat MPI* increases significantly, as number of cores (domains) increases. On the contrast, increasing for hybrid parallel programming model is not so

significant. Especially, that of HB 16×1 stays almost constant between 16 and 1,024 cores. Figure 13(b) provides parallel performance of T2K/Tokyo based on the performance of *flat MPI* with 16 cores. At 1,024 cores, parallel performance is approximately 60% of the performance at 16 cores. Decreasing of parallel performance of HB 16×1 is very significant. At 1,024 cores, HB 16×1 is rather slower than *flat MPI* although convergence is much better.

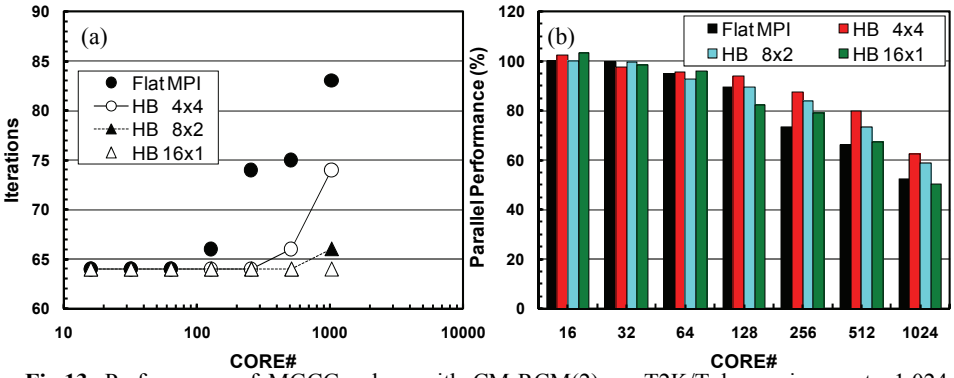


Fig.13. Performance of MGCG solver with CM-RCM(2) on T2K/Tokyo using up to 1,024 cores, Strong Scaling: 33,554,432 cells (=512×256×256), (a) Number of iterations for convergence, (b) Parallel performance based on the performance of *Flat MPI* with 16 cores

```
!C
!C-- SEND
      do neib= 1, NEIBPETOT
        istart= levEXPORT_index(lev-1,neib) + 1
        iend  = levEXPORT_index(lev  ,neib)
        inum  = iend - istart + 1
!$omp parallel do private (ii)
        do k= istart, iend
          WS(k)= X(EXPORT_ITEM(k))
        enddo
!$omp end parallel do
        call MPI_ISEND (WS(istart), inum, MPI_DOUBLE_PRECISION,           &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD, req1(neib), ierr)
      enddo
```

Fig.14. Communications for information exchange at domain boundary (sending process), copies of arrays to/from sending/receiving buffers occur

Communication between partitioned domains at each level occurs in parallel iterative solvers. Information at each domain boundary is exchanged using functions of MPI for point-to-point communication. In this procedure, copies of arrays to/from sending/receiving buffers occur, as shown in Fig.14. In the original code using OpenMP/MPI hybrid parallel programming models, this type of operation for memory copy is parallelized by OpenMP, as shown in Fig.14. But overhead of OpenMP is significant, if length of loop is short and number of threads is large. If length of loop is short, operations by a single thread might be faster than those by multi-threading.

In the current work, effect of switching from multi-threading to single-threading at coarser levels of multigrid procedure has been evaluated. Figure 15 (a) shows results of HB 16×1 with 1,024 cores (64 nodes) for the strong scaling case. "Communication" part includes processes of memory copies shown in Fig.14. "*Original*" applies multi-threading by OpenMP at every level of multigrid procedure. "*LEVcri=k*" means applying multi-threading if *level* of grid is smaller than *k*.

Therefore, single-threading is applied at every level if "*LEVcri=1*", and multi-threading is applied at only the finest grid (level=1) if "*LEVcri=2*". Generally speaking, "*LEVcri=2*" provides the best performance at 1,024 cores for all of HB 4×4, HB 8×2 and HB 16×1, although effect of switching is not so clear for HB 4×4, as shown in Fig.15 (b). Figure 16 shows effects of this optimization with "*LEVcri=2*" for all OpenMP/MPI hybrid cases in Fig.14 (b). Performance of HB 8×2 and HB 16×1 are much improved at large number of cores, and HB 8×2 is even faster than HB 4×4 at 1,024 cores, while performance with fewer number of cores did not change. Finally, performance of strong scaling has been evaluated between 16 and 1,024 cores of T2K/Tokyo and Cray-XT4. Each of Fig.17 (a) and (b) shows parallel speed-up to 1,024 cores with "*LEVcri=2*", based-on the performance of *flat MPI* with 16 cores of each platform. Performance at 1,024 cores for T2K/Tokyo is 534 (flat MPI), 690 (HB 4×4), 696 (HB 8×2), and 646 (HB 16×1), respectively. Performance of Cray-XT4 is 455 (flat MPI) and 617 (HB 4×4).

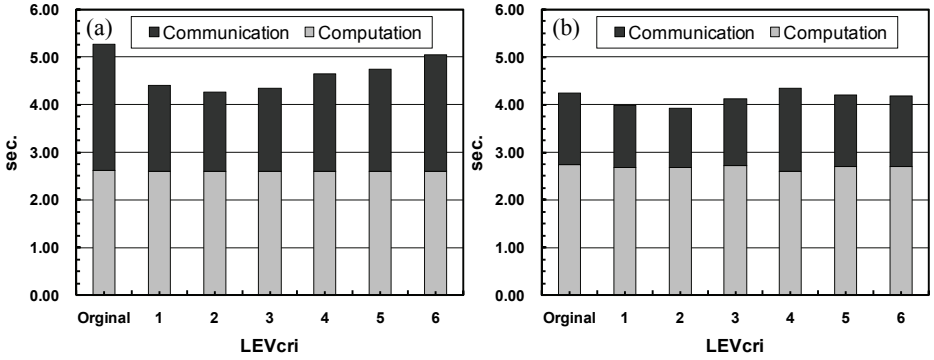


Fig.15. Effect of switching from multi-threading to single-threading at coarse levels of multigrid procedure in operations of memory copy for communications at domain boundaries using 1,024 cores for strong scaling case with 33,554,432 cells (=512×256×256), "*LEVcri=k*": applying multi-threading if level of grid is smaller than *k*, (a) HB 16×1, (b) HB 4×4

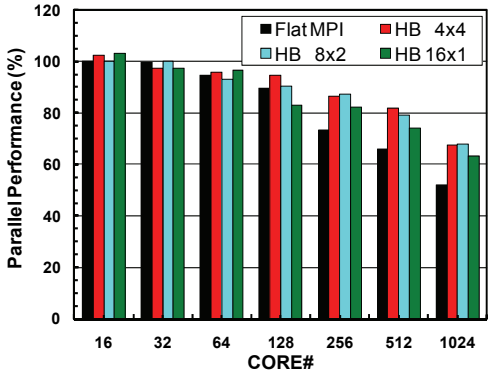


Fig.16. Performance of MGCG solver with CM-RCM(2) on T2K/Tokyo using up to 1,024 cores, Strong Scaling: 33,554,432 cells (=512×256×256), Parallel performance based on the performance of *Flat MPI* with 16 cores, "*LEVcri=2*" in Fig.15 is applied for OpenMP/MPI hybrid parallel programming models

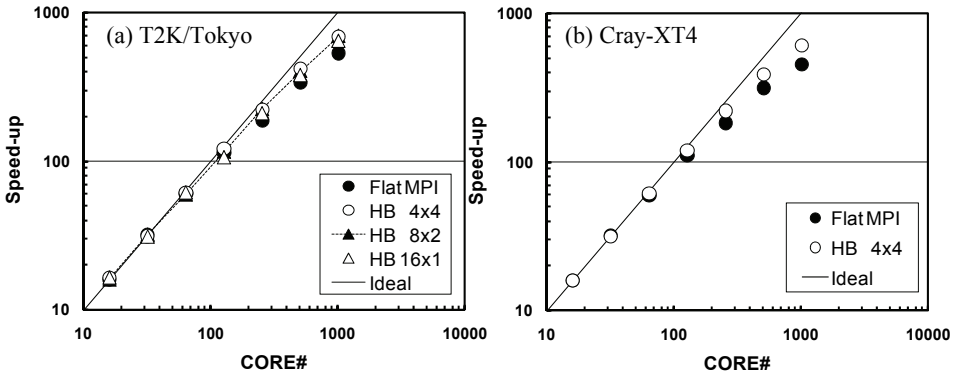


Fig.17. Performance of MGCG solver with CM-RCM(2) on T2K/Tokyo and Cray-XT4 using up to 1,024 cores, Strong Scaling: 33,554,432 cells (=512×256×256), Speed-up based on the performance of *flat MPI* with 16 cores on each platform, "*LEVcri=2*" in Fig.15 is applied for OpenMP/MPI hybrid parallel programming models

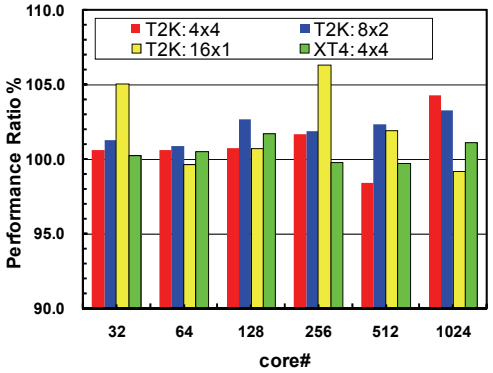


Fig.18. Performance of MGCG solver with CM-RCM(2) on T2K/Tokyo and Cray-XT4 using up to 1,024 cores, Weak Scaling: 262,144 cells/core, Max. Total Problem Size: 268,435,456, Ratio of performance for the solver with "*LEVcri=2*" to that of original solver in Fig.12

Finally, optimized solver for strong scaling cases with "*LEVcri=2*" has been applied to weak scaling cases, where number of finite-volume cells per each core is 262,144 (=64^3). Figure 18 shows ratio of performance for OpenMP/MPI hybrid cases up to 1,024 cores. Generally speaking, entire performance is not so much changed by optimization for strong scaling cases. Therefore, switching from multi-threading to single-threading at coarser levels of multigrid procedure in operations of memory copy for communications at domain boundaries, works well for both of weak and strong scaling cases.

5 Concluding Remarks

OpenMP/MPI hybrid parallel programming models were implemented to 3D finite-volume based simulation code for groundwater flow problems through heterogeneous

porous media using parallel iterative solvers with multigrid preconditioning by IC(0) smoothing. Performance and robustness of the developed code has been evaluated on T2K/Tokyo and Cray-XT4 using up to 1,024 cores through both of weak and strong scaling computations. Optimization procedures for OpenMP/MPI hybrid parallel programming models, such as appropriate command lines for NUMA control, first touch data placement and reordering for contiguous "sequential" access to memory, provided excellent improvement of performance on multigrid preconditioners. Furthermore, performance of OpenMP/MPI hybrid at large number of cores in strong scaling is improved by optimization of communication procedure between domains. The developed procedure also provided good performance in weak scaling cases. OpenMP/MPI hybrid demonstrated better performance and robustness than *flat MPI*, especially with large number of cores for ill-conditioned problems, and could be a reasonable choice for large-scale computing on multi-core/multi-socket clusters. Automatic selection of optimum parallel programming models and parameters (e.g. number of colors, switching level for communication) is an interesting area for future works. Furthermore, development of parallel procedures for reordering and more robust procedures for parallel multigrid using HID (Hierarchical Interface Decomposition) [1] is also ongoing. In the current work, the developed procedures were evaluated under very limited conditions. Various geometries, boundary conditions and problem size will be applied in the future for various types of computer platforms.

References

1. Nakajima, K.: Flat MPI vs. Hybrid: Evaluation of Parallel Programming Models for Preconditioned Iterative Solvers on "T2K Open Supercomputer", IEEE Proceedings of the 38th International Conference on Parallel Processing (ICPP-09), pp.73-80 (2009)
2. Information Technology Center, The University of Tokyo: http://www.cc.u-tokyo.ac.jp/
3. Nakajima, K.: Parallel Multilevel Method for Heterogeneous Field, IPSJ Proceedings of HPCS 2006, pp.95-102 (2006) (in Japanese)
4. NERSC, Lawrence Berkeley National Laboratory: http://www.nersc.gov/
5. The T2K Open Supercomputer Alliance: http://www.open-supercomputer.org/
6. Deutsch, C.V., Journel, A.G.: GSLIB Geostatistical Software Library and User's Guide, Second Edition. Oxford University Press (1998)
7. Tottemberg, U., Oosterlee, C. and Schuller, A.: Multigrid, Academic Press (2001)
8. Nakajima, K.: Parallel Multilevel Iterative Linear Solvers with Unstructured Adaptive Grids for Simulations in Earth Science, Concurrency and Computation: Practice and Experience 14-6/7, pp.484-498 (2002)
9. Smith, B., Bjφrstad, P. and Gropp, W. : Domain Decomposition, Parallel Multilevel Methods for Elliptic Partial Differential Equations, Cambridge Press (1996)
10. Mattson, T.G., Sanders, B.A., Massingill, B.L.: Patterns for Parallel Programming, Software Patterns Series (SPS), Addison-Wesley (2005)
11. Washio, T., Maruyama, K., Osoda, T., Shimizu, F., Doi, S.: Efficient implementations of block sparse matrix operations on shared memory vector machines. Proceedings of The 4th International Conference on Supercomputing in Nuclear Applications (SNA2000) (2000)
12. STREAM (Sustainable Memory Bandwidth in High Performance Computers): http://www.cs.virginia.edu/stream/