Automatic Tuning of Parallel Multigrid Solvers Using OpenMP/MPI Hybrid Parallel Programming Models

Kengo Nakajima

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

Abstract. The multigrid method with OpenMP/MPI hybrid parallel programming model is expected to play an important role in large-scale scientific computing on post-peta/exa-scale supercomputer systems. Because the multigrid method includes various choices of parameters, selecting the optimum combination of these is a critical issue. In the present work, we focus on the selection of single-threading or multi-threading in the procedures of parallel multigrid solvers using OpenMP/MPI parallel hybrid programming models. We propose a simple empirical method for automatic tuning (AT) of related parameters. The performance of the proposed method is evaluated on the T2K Open Supercomputer (T2K/Tokyo), the Cray XE6, and the Fujitsu FX10 using up to 8,192 cores. The proposed method for AT is effective, and the automatically tuned code provides twice the performance of the original one.

Keywords: Multigrid, Hybrid Parallel Programming Model, Automatic Tuning.

1 Introduction

To achieve minimal parallelization overheads on 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, and fine-grained parallelism is obtained via loop-level parallelism inside each node by using compiler-based thread parallelization techniques such as OpenMP. Another often used programming model is the single-level *flat MPI* model, in which separate single-threaded MPI processes are executed on each core.

In previous works [1,2], OpenMP/MPI hybrid parallel programming models were implemented in 3D finite-volume simulation code for groundwater flow problems through heterogeneous porous media using parallel iterative solvers with multigrid preconditioning. The performance and the robustness of the developed code was evaluated on the T2K Open Supercomputer at the University of Tokyo (T2K/Tokyo) [3,4] using up to 8,192 cores for both weak and strong scaling computations. Furthermore, a new strategy for solving equations at the coarsest level (coarse grid solver) was proposed and evaluated in [2], and the new coarse grid solver improved

© Springer-Verlag Berlin Heidelberg 2013

the scalability of the multigrid solver dramatically. The OpenMP/MPI hybrid parallel programming model, in which one MPI process was applied to a single quad-core socket of the T2K/Tokyo with four OpenMP threads (HB 4×4) [1,2], demonstrated the best performance and robustness for large-scale ill-conditioned problems by appropriate optimization and coarse grid solvers. In [5], performance of parallel programming models for algebraic multigrid solvers in Hypre Library [6] have been evaluated on various multicore HPC platforms with more than 10^5 cores, such as IBM BlueGene/P, and Cray XT5. The *MultiCore SUPport library (MCSup)* [5] provides a framework, in which the optimization processes described in [1,2] are applied automatically. Results show that threads of an MPI process should always be kept on the same socket for optimum performance to achieve both memory locality and to minimize OS overhead for cc-NUMA architecture. This corresponds to HB 4×4 programming model in [1,2].

The concepts of OpenMP/MPI hybrid parallel programming models can be easily extended and applied to supercomputers based on heterogeneous computing nodes with accelerators/co-processors, such as GPUs and/or many-core processors by Intel Many Integrated Core Architecture. Multigrid is a scalable method for solving linear equations and for preconditioning Krylov iterative linear solvers, and it is especially suitable for large-scale problems. The multigrid method is expected to be one of the powerful tools on post-peta/exa-scale systems. It is well known that the multigrid method includes various choices of parameters. Because each of these strongly affects the accuracy, the robustness, and the performance of multigrid procedures, selection of the optimum combination of these is very critical. In OpenMP/MPI hybrid parallel programming models, the number of threads strongly affects the performance of both the computation and the communications in multigrid procedures [1].

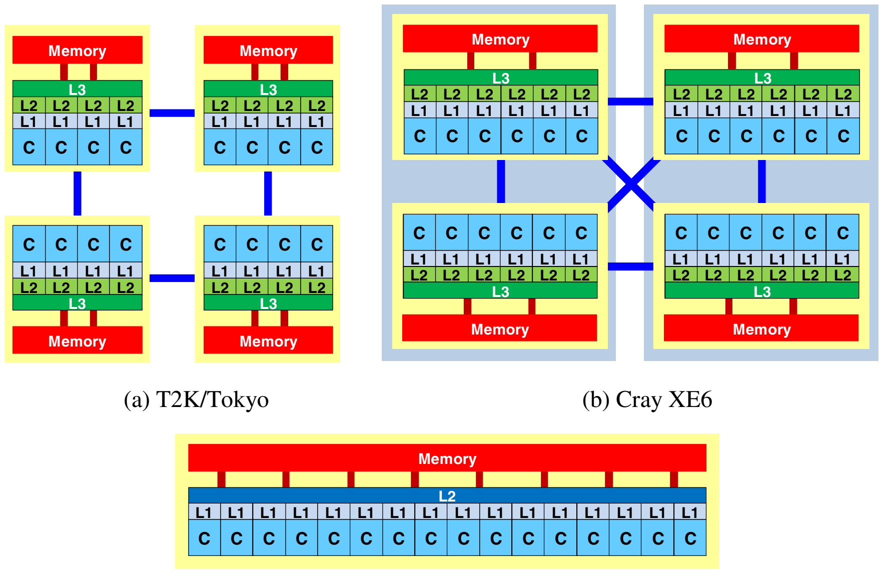
In the present work, we focus on the selection of single-threading or multi-threading in procedures of parallel multigrid solvers using OpenMP/MPI hybrid parallel programming models. We propose a new method of automatic tuning (AT) of the parameters. The proposed method implemented in the code in [2] is evaluated by using up to 8,192 cores of the T2K/Tokyo, the Cray XE6 [7], and the Fujitsu FX10 [3]. The rest of this paper is organized as follows. In Section 2, an overview of the target hardware is provided. In Section 3, we outline the target application and give a summary of the results in [1] and [2]. In Section 4, details of our new method for automatic tuning and the results of the computations are described, while some final remarks are offered in Sections 5.

2 Hardware Environment

Table 1 and Fig. 1 summarize features of the architectures of the three target systems used in the present work. The T2K/Tokyo was developed by Hitachi under the “T2K Open Supercomputer Alliance” [4]. It is a combined cluster system with 952 nodes, 15,232 cores and 31 TB memory. The total peak performance is 140 TFLOPS. Each node includes four sockets of AMD quad-core Opteron (Barcelona) processors (2.3 GHz), as shown in Fig. 1(a). Each socket is connected through HyperTransport links, and the computing nodes are connected via a Myrinet-10G network, which has a multi-stage cross-bar network topology. In the present work, 512 nodes of the system are evaluated. Because T2K/Tokyo is based on *cache-coherent* NUMA (cc-NUMA) architecture, careful design of both the software and the data configuration is required for efficient access to local memory.

Table 1. Summary of specifications: Computing node of the target systems

	T2K/Tokyo	**Cray XE6**	**Fujitsu FX10**
Core #/Node	16	24	16
Size of Memory/node (GB)	32	32	32
Peak Performance/node (GFLOPS)	147.2	201.6	236.5
Peak Memory Bandwidth/node (GB/sec)	42.7 8×DDR2 667MHz	85.3 8×DDR3 1333MHz	
STREAM/Triad Performance/node (GB/sec) [8]	20.0	52.3	64.7
B/F Rate	0.136	0.260	0.274



(a) T2K/Tokyo

(b) Cray XE6

(c) Fujitsu FX10

Fig. 1. Overview of a computing node of (a) the T2K/Tokyo, (b) the Cray XE6, and (c) the Fujitsu FX10 (C: core, L1/L2/L3: cache, Memory: main memory)

Each node of the Cray XE6 (Hopper) system at the National Energy Research Scientific Computing Center (NERSC), Lawrence Berkeley National Laboratory [7] includes two sockets of 12-core AMD Opteron (Magny-Cours) processors (2.1 GHz). Each socket of the Magny-Cours consists of two *dies*, each of which consists of six cores. Four dies are connected through HyperTransport links (Fig. 1(b)). The Magny-Cours has more HyperTransport links than the Barcelona processor, and the four dies are connected more tightly. An entire system consists of 6,384 nodes, 153,216 cores, and 212 TB memory. The total peak performance is 1.28 PFLOPS. The computing nodes are connected via Cray's Gemini network, which has a 3D torus network topology. In the present work, 128 nodes of the system are evaluated. Both the T2K/Tokyo and the Cray XE6 are based on cc-NUMA architecture. Each die with six cores of the Cray XE6 corresponds to a socket with four cores of the T2K/Tokyo.

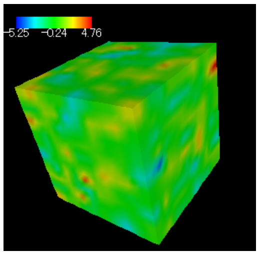
The Fujitsu FX10 (Oakleaf-FX) system at the University of Tokyo [3] is Fujitsu's PRIMEHPC FX10 massively parallel supercomputer with a peak performance of 1.13 PFLOPS. The Fujitsu FX10 consists of 4,800 computing nodes of SPARC64™ IXfx processors with 16 cores (1.848 GHz). SPARC64™ IXfx incorporates many features for HPC, including a hardware barrier for high-speed synchronization of on-chip cores [3]. An entire system consists of 76,800 cores and 154 TB memory. Nodes are connected via a six-dimensional mesh/torus interconnect called "Tofu" [3]. In the present work, 128 nodes of the system are evaluated. On the SPARC64™ IXfx, each of the 16 cores can access the memory in a uniform manner (Fig. 1(c)).

3 Algorithms and Implementations of the Target Application

3.1 Overview of the Target Application

In the target application, Poisson's equations for groundwater flow problems through heterogeneous porous media are solved using a parallel *cell-centered* 3D finite-volume method (FVM) (Fig.2) [1,2]. A heterogeneous distribution of water conductivity in each mesh is calculated by a sequential Gauss algorithm [9]. The minimum and the maximum values of water conductivity are 10^{-5} and 10^{5}, respectively, and the average value is 1.0. This configuration provides ill-conditioned coefficient matrices whose condition number is approximately 10^{10}. Each mesh is a cube, and the distribution of meshes is structured as finite-difference-type voxels.

(a)



(b)

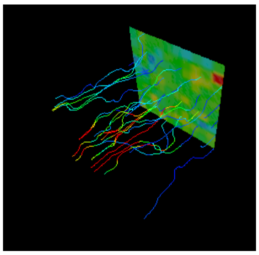


Fig. 2. Example of groundwater flow through heterogeneous porous media (a) Distribution of water conductivity, (b) Streamlines

The conjugate gradient (CG) solver with a multigrid preconditioner (MGCG) is applied for solving the Poisson's equations [1,2]. A very simple geometric multigrid with a V-cycle, where eight children form one parent mesh in an isotropic manner for structured finite-difference-type voxels, is applied [1,2]. The *level* of the finest grid is set to 1 and the level is numbered from the finest to the coarsest grid, where the number of meshes is one in each domain (MPI process). Multigrid operations at each level are done in parallel manner, but the operations at the coarsest levels are executed on a single MPI process by gathering the information of entire processes. The total number of meshes at the coarsest level is equal to the number of MPI processes. IC(0) with additive Schwarz domain decomposition (ASDD) [1,2], because of its robustness, is adopted as the smoothing operator at each level. The 3D code is parallelized by domain decomposition using MPI for communications between partitioned domains [1,2]. 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 to achieve the optimum parallel performance of IC operations in multigrid processes. In the present work, Reverse Cuthill-McKee (RCM) with cyclic-multicoloring (CM-RCM) [10] is applied. The number of colors is set to 2 at each level (CM-RCM(2)) for efficiency [1,2]. The following three types of optimization procedures for cc-NUMA architectures are applied to the OpenMP/MPI hybrid parallel programming models [1,2]:

- Appropriate command lines for NUMA control, with “`--cpunodebind`” and “`--localalloc`”, where memory locality is kept, and each thread can access data on the memory of each socket efficiently [1]
- First touch data placement [1,2]
- Reordering for contiguous “sequential” access to memory [1]

Furthermore, optimization of the coarse grid solver proposed in [2] is applied.

3.2 Results (Weak Scaling)

The performance of weak scaling is evaluated using between 16 and 8,192 cores of the T2K/Tokyo. The number of finite-volume meshes per each core is 262,144 (=64^3); therefore, the maximum total problem size is 2,147,483,648. The following three types of OpenMP/MPI hybrid parallel programming models are applied as follows, and the 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(a), four MPI processes in each node
- **Hybrid 8×2 (HB 8×2):** Eight OpenMP threads for two pairs of sockets, two MPI processes in each node
- **Hybrid 16×1 (HB 16×1):** Sixteen OpenMP threads for a single node, one MPI process in each node

In Fig. 3, (a) and (b) show the performance of the MGCG solver. An improved version of the coarse grid solver (C2) proposed in [2] is applied, where a multigrid based on the V-cycle with IC(0) smoothing is applied until convergence ($\varepsilon=10^{-12}$) at the coarsest level [2]. Both figures show the scalable features of the developed method. The number of iterations until convergence and the elapsed time for MGCG solvers at 8,192 cores are as follows:

• **Flat MPI:**	70 iterations, 35.7 sec.
• **HB 4×4:**	71 iterations, 28.4 sec.
• **HB 8×2:**	72 iterations, 32.8 sec.
• **HB 16×1:**	72 iterations, 34.4 sec.

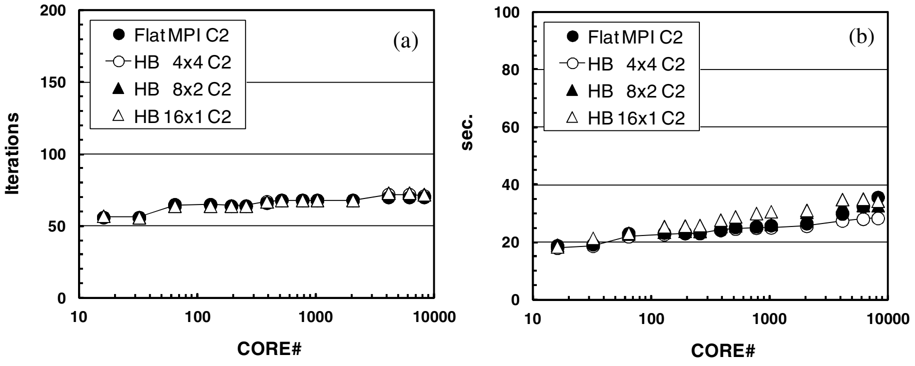


Fig. 3. Performance of MGCG solver with CM-RCM(2) on the T2K/Tokyo using up to 8,192 cores, weak scaling: 262,144 meshes/core, maximum total problem size: 2,147,483,648. (a) Number of iterations for convergence, (b) Computation time for MGCG solvers with improved coarse grid solver (C2) applied.

MGCG is a *memory-bound* process, and the performance of memory access is very critical. The performance of HB 4×4 is the best, primarily because all data for each process are guaranteed to be on the local memory of each socket, and so the most efficient memory access is possible. HB 4×4 is the best according to both the elapsed computation time and the performance in a single iteration [2]. Flat MPI is also better than the others for a small number of cores, but it consists of a larger number of MPI processes than OpenMP/MPI hybrid parallel programming models. Moreover, the problem size for the coarse grid solver is larger than that of these hybrid parallel programming models. Therefore, its performance gets worse for a larger number of cores due to the overhead of communications and coarse grid solvers.

3.3 Results (Strong Scaling) and the Optimization of Communication

The performance of strong scaling is evaluated for a fixed size of problem with 33,554,432 meshes (=512×256×256) using between 16 and 1,024 cores of the T2K/Tokyo [1]. Figure 4(a) provides the parallel performance of the T2K/Tokyo based on the performance of flat MPI with 16 cores using the original coarse grid solver in [1]. At 1,024 cores, the parallel performance is approximately 60% of the performance at 16 cores. Decreasing of the parallel performance of HB 16×1 is very significant. At 1,024 cores, HB 16×1 is slower than flat MPI, although the convergence is much better [1]. Communications between partitioned domains at each level occur in the parallel multigrid procedures. Information at each domain boundary is exchanged by using the functions of MPI for point-to-point communications. In this procedure, copies of arrays to/from sending/receiving buffers are made, as shown in Fig. 5. In the original code using OpenMP/MPI hybrid parallel programming models, this type of operation for the memory copy is parallelized by OpenMP. But the overhead of OpenMP is significant if the length of the loop is short at the coarser levels of the multigrid procedure and the number of threads is large. If the length of the loop is short, operations by a single thread might be faster than those by multi-threading.

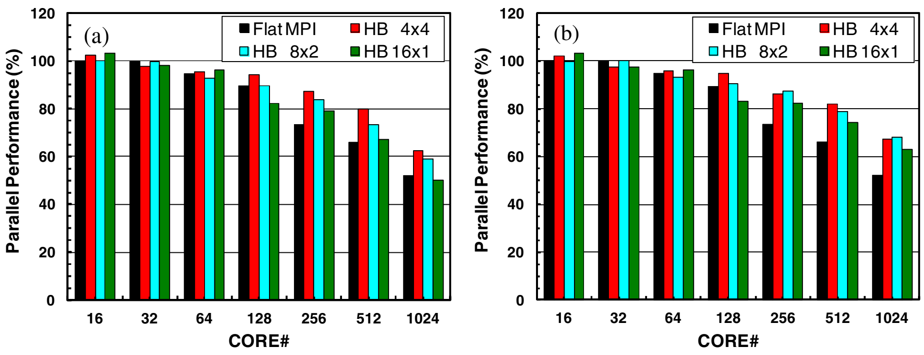


Fig. 4. Performance of MGCG solver with CM-RCM(2) on the T2K/Tokyo using up to 1,024 cores, strong scaling: 33,554,432 meshes (=512×256×256). (a) Parallel performance based on the performance of flat MPI with 16 cores, (a) Initial case, (b) Optimized case: "*LEVcri=2*" in Fig. 7 is applied for OpenMP/MPI hybrid parallel programming models [1].

```
!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, SOLVER_COMM, req1(neib), ierr)
      enddo
```

Fig. 5. Point-to-point communications for information exchange at the domain boundary (sending process), copies of arrays to/from sending/receiving buffers occur

In [1], the effect of switching from multi-threading to single-threading at coarser levels of the multigrid procedure was evaluated. Figure 6(a) shows the results of HB 16×1 with 1,024 cores (64 nodes) for the strong scaling case. The "communication" part includes processes of the memory copies shown in Fig. 5. "*LEVcri=0*" is the original case, and it applies multi-threading by OpenMP at every level of the multigrid procedure. "*LEVcri=k* (*k*>0)" means applying multi-threading if the level of the 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, "*LEVcri=2*" provides the best performance at 1,024 cores for all of HB 4×4, HB 8×2, and HB 16×1. The optimized HB 16×1 with "*LEVcri=2*" is 22% faster than that of the original case, although the effect of switching is not so clear for HB 4×4. Figure 4(b) shows the effects of this optimization with "*LEVcri=2*" for all OpenMP/MPI hybrid cases. The performance of HB 8×2 and HB 16×1 are much improved at a large number of cores, and HB 8×2 is even faster than HB 4×4 at 1,024 cores, while the performance with a fewer number of cores does not change.

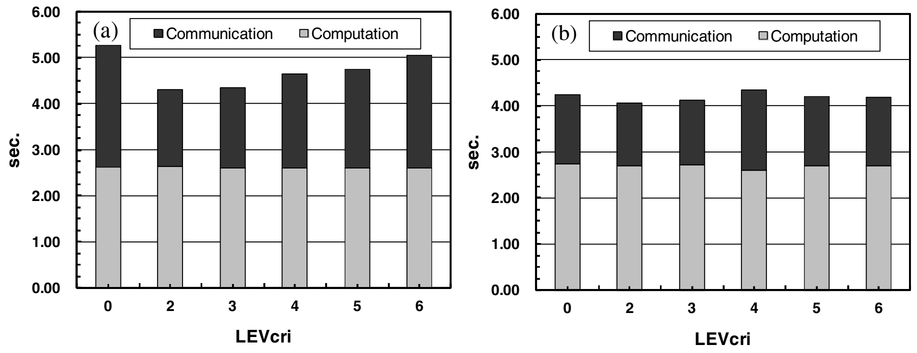


Fig. 6. Effect of switching from multi-threading to single-threading at coarse levels of the multigrid procedure in operations of memory copy for communications at domain boundaries using 1,024 cores for a strong scaling case with 33,554,432 meshes (=512×256×256), "*LEVcri=0*": applying multi-threading by OpenMP at every level of the multigrid procedure (original case), "*LEVcri=k (k>*0*)*": applying multi-threading if the grid level is smaller than *k*. (a) HB 16×1, (b) HB 4×4.

4 Automatic Tuning (AT) of Multigrid Processes

4.1 Overview

In multigrid procedures with OpenMP/MPI hybrid parallel programming models, most of the processes are parallelized by OpenMP at each level. But the overhead of OpenMP is significant if the length of the loop is short at coarser levels and the number of threads is large. If the length of the loop is short, operations by a single thread might be faster than those by multi-threading, as shown in 3.3. In 3.3, the optimum parameter ("*LEVcri=2*") is determined by comparing the results of cases with different values of *LEVcri* between 0 and 6. But this optimum parameter depends on various conditions, such as problem size, number of processors, number of threads per each MPI process, architecture of hardware, performance of computing nodes, communication performance of network, etc. Therefore, automatic tuning (AT) is helpful for the selection of the optimum parameters.

4.2 Method for Automatic Tuning (AT)

In the present work, we focus on the selection of single-threading or multi-threading in procedures of parallel multigrid solvers using OpenMP/MPI hybrid parallel programming models. The method for AT of related parameters was proposed, and was implemented to the code in [2], which was optimized for cc-NUMA architectures, such as the T2K/Tokyo and the Cray XE6. Finally, the proposed method is evaluated using the three supercomputer systems. In the present work, the following three types of parallel programming models are evaluated:

- **Hybrid 4×4/6×4 (HB 4×4/6×4):** Four MPI processes on each node. Four OpenMP threads/MPI process for the T2K/Tokyo and the Fujitsu FX10, six threads/MPI process for the Cray XE6
- **Hybrid 8×2/12×2 (HB 8×2/12×2):** Two MPI processes on each node. Eight OpenMP threads/MPI process for the T2K/Tokyo and the Fujitsu FX10, 12 threads/MPI process for the Cray XE6
- **Hybrid 16×1/24×1 (HB 16×1/24×1):** One MPI process in each node. Number of threads corresponds to the number of cores on each node (16: T2K/Tokyo, Fujitsu FX10, 24: Cray XE6)

We focus on the automatic selection of single-threading or multi-threading in the following three procedures of parallel multigrid solvers using the OpenMP/MPI hybrid parallel programming models:

(A) Smoothing operations at each level of the V-cycle
(B) Point-to-point communications at domain boundaries with the memory copies described in 3.3
(C) Smoothing operations at each level of the coarse grid solver

Process (A) corresponds to smoothing operations for each MPI process at each level of the V-cycle, whereas process (C) corresponds to smoothing operations at each level of the coarse grid solver on a single MPI process [2]. The level of the multigrid for switching from multi-threading to single-threading is defined as $LEVcri_A$ for process (A), $LEVcri_B$ for process (B), and $LEVcri_C$ for process (C). The definition of $LEVcri_X$ is the same as that of *LEVcri* in 3.3 [1]. If "$LEVcri_X$=0", multi-threading is applied at every level of the process (X). "$LEVcri_X=k$ (k>0)" means multi-threading is applied to the process (X) if the level of the grid is smaller than k. The *policy* for optimization is defined as a combination of these three parameters ($LEVcri_A$, $LEVcri_B$, and $LEVcri_C$). In the present work, this policy is represented by a three-digit number, where each digit corresponds to each $LEVcri_X$. For example, the policy represented by "542" means "$LEVcri_A=5$", "$LEVcri_B=4$", and "$LEVcri_C=2$". In the present work, we develop a method for AT that can automatically define the optimum policy (*i.e.*, the combination of optimum $LEVcri_X$'s) for various kinds of hardware and software conditions. $LEVcri_A$ and $LEVcri_C$ are defined by a critical loop length *LOOPcri*, which is a parameter for selection of single-threading or multi-threading, and is calculated by the simple *off-line* benchmark shown in Fig. 7. This benchmark simulates typical and costly processes in smoothing operations, such as sparse-matrix-vector products and forward/backward substitutions for IC(0) operations [1,2]. Six off-diagonal components are used because the target application is based on cell-centered structured 3D meshes with six surfaces. This off-line benchmark compares the performance of loops with single-threading and multi-threading for various loop lengths, *N*, and automatically introduces the critical loop length *LOOPcri*. *LOOPcri* is a function of the number of threads, the computational performance of each core and the memory bandwidth. Table 2 shows the *LOOPcri* calculated by a single node of each supercomputer system for each parallel programming model. *LOOPcri* of the FX10 is smaller because of its hardware barrier for high-speed synchronization of on-chip cores. Generally, this off-line benchmark needs to be performed just once, as long as computational environment, such as version of the compiler, does not change significantly. We just provide *LOOPcri* as one of the input parameters of the application, in which the proposed method for AT is implemented. If the loop length is larger than *LOOPcri*, multi-threading is applied. Optimization of process (B) (*i.e.*, selection of $LEVcri_B$) is done by a run-time tuning procedure. This run-time tuning procedure is embedded as one of the subroutines of the target application written in FORTRAN. This subroutine (**`comm_test`**) is called by the main program of the application before starting of the real computations. This subroutine (**`comm_test`**) compares the performance of single-threaded and multi-threaded versions of communication functions at each level, and chooses the faster one for the real computations, as shown in Fig. 8. This procedure is very convenient and reliable because it can evaluate the combined performance for both MPI communication and memory copying by the real functions used in MGCG solvers of the target application. Moreover, this run-time tuning procedure is not costly: it takes less than 0.05 sec. for all cases in the present work. Finally, Figure 9 summarizes the procedure of proposed method for AT.

```
!$omp parallel do private (i,k)
      do i= 1, N
        Y(i)= D(i)*X(i)
        do k= 1, 6
          Y(i)= Y(i) + A(k,i)*X(i)
        enddo
      enddo
!$omp end parallel do

      do i= 1, N
        Y(i)= D(i)*X(i)
        do k= 1, 6
          Y(i)= Y(i) + A(k,i)*X(i)
        enddo
      enddo
```

Fig. 7. Off-line benchmark, which defines critical loop length *LOOPcri* for the selection of single-threading/multi-threading. If loop length *N* is larger than *LOOPcri*, multi-threading is applied.

Table 2. LOOPcri measured by the simple off-line benchmark in Fig. 7

	T2K/Tokyo	**Cray XE6**	**Fujitsu FX10**
HB 4×4/ 6×4	256	64	32
HB 8×2/12×2	512	128	32
HB 16×1/24×1	1,024	256	32

1. At each *level* of V-cycle, execution time *T(m,level)* for point-to-point communications at domain boundaries (including memory copies and MPI communications) with multi-threading using OpenMP is measured.
2. At each *level* of V-cycle, execution time *T(s,level)* for point-to-point communications at domain boundaries (including memory copies and MPI communications) with single-threading (without OpenMP) is measured.
3. Each of 1. and 2. is repeated for 50 times at each level, and average execution time, $T_{ave}(m,level)$ and $T_{ave}(s,level)$ are calculated.
4. If $T_{ave}(m,level) > T_{ave}(s,level)$, single-threading is adopted at this level of V-cycle. Otherwise, multi-threading is adopted.
5. Optimum $LEVcri_B$ is determined through these procedures.

Fig. 8. Procedure of **`subroutine comm_test`** for run-time tuning for optimization of process (B) (point-to-point communications at domain boundaries with the memory copies)

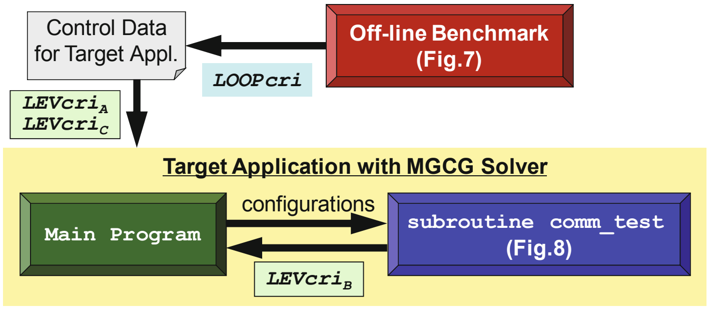

Control Data for Target Appl.
Off-line Benchmark (Fig.7)
LOOPcri
LEVcri_A
LEVcri_C
Target Application with MGCG Solver
Main Program
configurations
subroutine comm_test (Fig.8)
LEVcri_B

Fig. 9. Procedures for the proposed method of AT

4.3 Preliminary Results

The method for AT proposed in 4.2 is implemented with the code in [2]. CM-RCM(2) reordering [1,2] is applied at each level; therefore, the loop length at each level is half of the problem size. The following three types of problem sizes are evaluated:

• Large:	2,097,152	(=128×128×128) meshes per each node
• Medium:	524,288	(=128×64×64) meshes per each node
• Small:	65,536	(=64×32×32) meshes per each node

Tables 3, 4, and 5 show the results for the three types of problem sizes on the T2K/Tokyo (512 nodes, 8,192 cores), the Cray XE6 (128 nodes, 3,072 cores), and the Fujitsu FX10 (128 nodes, 2,048 cores), respectively. For each case, two types of codes are developed and applied.

The first code is based on the method for AT described in 4.2, and Fig.9. This code is implemented so that critical loop length *LOOPcri* in Table 2 provides the optimum $LEVcri_A$ and $LEVcri_C$ automatically, while the run-time tuning procedure by a subroutine (`comm_test`) of the first code described in Fig. 8 provides the optimum $LEVcri_B$ automatically. The rows with "*AT*" in Tables 3, 4, and 5 show the results of this first code and provide the speed-up compared to that of the original case (policy=”000”). The three-digit numbers in parentheses are the policy for optimization provided by AT.

In the second code, each of $LEVcri_A$, $LEVcri_B$, and $LEVcri_C$ can be explicitly specified, where two of these parameters are fixed as “0” in the present work. According to the results by the second code, the best combination of parameters can be estimated. The rows with “*Estimated Best*” in Tables 3, 4, and 5 show the results of the second code (speed-up and corresponding policy for optimization). Figures 10, 11, and 12 show the effect of the individual parameter. The performance is 1.00 for the original case (policy=”000”).

The effect of tuning by switching from multi-threading to single-threading is significant on the T2K/Tokyo and the Cray XE6 if the problem size per node is small and the number of threads per node is large. Generally, “AT” provides better performance than “Estimated Best” in each case; therefore, the AT procedure works well. The optimum policies (*i.e.*, combinations of optimum parameters) provided by “AT” and “Estimated Best” in Tables 3, 4, and 5 are similar. The “small” size cases for the T2K/Tokyo with HB 16×1, and the Cray XE6 with HB 24×1 provide 1.75–1.96 times speed-up by AT, compared to the performance of the original cases. In contrast, the effect of this type of tuning is very small for the “large” size cases. Among the three parameters ($LEVcri_A$, $LEVcri_B$, and $LEVcri_C$), the effect of $LEVcri_B$ is the most critical in the “small” and “medium” size cases. Therefore, accurate estimation of the optimum $LEVcri_B$ is important. The optimum value of the parameter varies according to the problem size. For example, “$LEVcri_B=2$ (020)” is the best for “medium” size cases (Fig. 10), while “$LEVcri_B=1$ (010)” provides the best performance for “small” size cases (Fig. 11). The effect of tuning by switching from multi-threading to single-threading is very small on the Fujitsu FX10 (Table 5 and Fig. 12). This is because of its hardware barrier for high-speed synchronization of on-chip cores. Multi-threading provides higher efficiency even for short loops on the Fujitsu FX10, as shown in Table 2. Finally, Table 6 compares the performance of the three types of OpenMP/MPI hybrid parallel programming models on three supercomputers for “small” size cases, where the effect of the proposed AT procedure is the most significant. Generally, HB 4×4/6×4 provides the best performance for original, except Fujitsu FX10. But HB 8×2 for T2K/Tokyo and Fujitsu FX10 optimized by the proposed AT procedure is even faster than optimized HB 4×4.

Table 3. Results on the T2K/Tokyo with 512 nodes (8,192 cores): Speed-up compared to the original case (policy=”000”) and policy for optimization (three-digit number in parentheses)

		Large	**Medium**	**Small**
HB 4×4	AT	1.02 (522)	1.04 (422)	1.08 (322)
	Estimated Best	1.01 (522)	1.03 (613)	1.14 (612)
HB 8×2	AT	1.01 (522)	1.10 (412)	1.40 (311)
	Estimated Best	1.02 (532)	1.08 (412)	1.33 (512)
HB 16×1	AT	1.03 (421)	1.22 (421)	1.96 (311)
	Estimated Best	1.02 (633)	1.10 (521)	1.89 (511)

Table 4. Results on the Cray XE6 with 128 nodes (3,072 cores): Speed-up compared to the original case (policy=”000”) and policy for optimization (three-digit number in parentheses)

		Large	**Medium**	**Small**
HB 6×4	AT	1.02 (522)	1.09 (422)	1.49 (322)
	Estimated Best	1.02 (642)	1.07 (432)	1.40 (612)
HB 12×2	AT	1.01 (632)	1.11 (521)	1.57 (311)
	Estimated Best	1.01 (641)	1.08 (621)	1.68 (421)
HB 24×1	AT	1.05 (531)	1.18 (531)	1.75 (321)
	Estimated Best	1.05 (641)	1.18 (541)	1.63 (421)

Table 5. Results on the Fujitsu FX10 with 128 nodes (2,048 cores): Speed-up compared to the original case (policy=”000”) and policy for optimization (three-digit number in parentheses)

		Large	**Medium**	**Small**
HB 4×4	AT	1.01 (532)	1.01 (532)	1.06 (422)
	Estimated Best	1.00 (053)	1.01 (023)	1.06 (622)
HB 8×2	AT	1.00 (532)	1.02 (532)	1.06 (422)
	Estimated Best	1.00 (042)	1.01 (542)	1.04 (522)
HB 16×1	AT	1.00 (042)	1.01 (642)	1.06 (422)
	Estimated Best	1.00 (052)	1.01 (030)	1.08 (532)

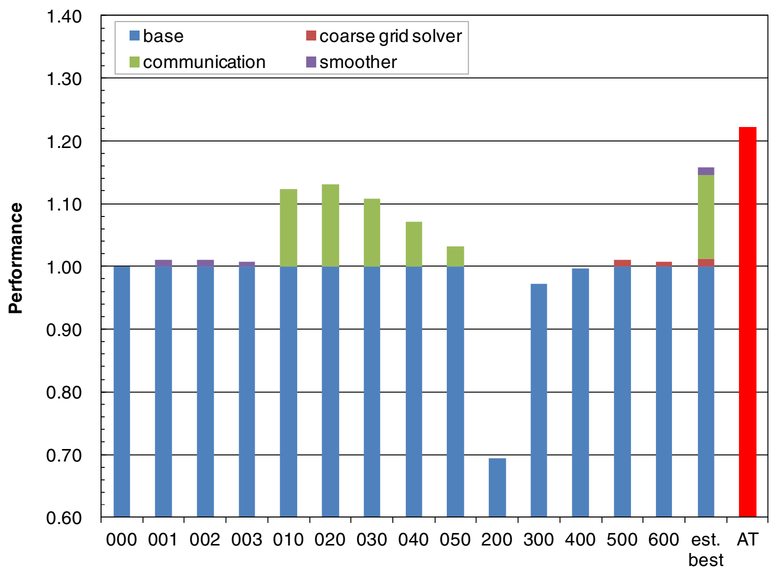


Fig. 10. Effect of individual parameters on speed-up: $LEVcri_A$ (smoother), $LEVcri_B$ (communication), and $LEVcri_C$ (smoother for coarse grid solver), T2K/Tokyo with 512 nodes, 8,192 cores, HB 16×1, “medium” size case (524,288 (=128×64×64) meshes/node)

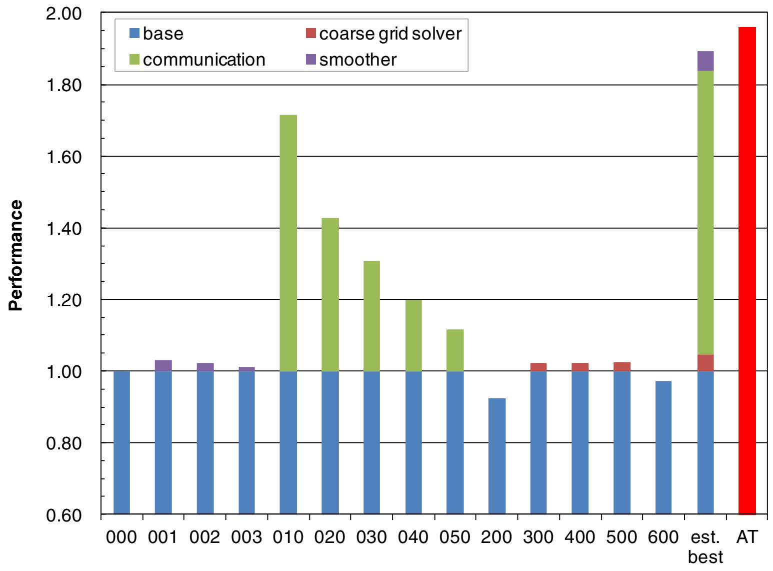


Fig. 11. Effect of individual parameters on speed-up: $LEVcri_A$ (smoother), $LEVcri_B$ (communication), and $LEVcri_C$ (smoother for coarse grid solver), T2K/Tokyo with 512 nodes, 8,192 cores, HB 16×1, “small” size case (65,536 (=64×32×32) meshes/node)

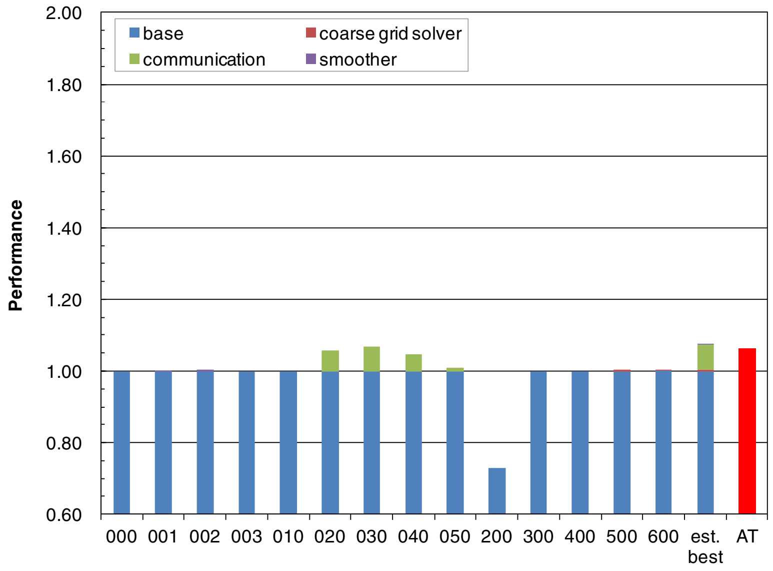


Fig. 12. Effect of individual parameters on speed-up: $LEVcri_A$ (smoother), $LEVcri_B$ (communication), and $LEVcri_C$ (smoother for coarse grid solver), Fujitsu FX10 with 128 nodes, 2,048 cores, HB 16×1, "small" size case (65,536 (=64×32×32) meshes/node)

Table 6. Speed-up compared to that of the original case (policy=000) of HB 4×4/6×4 for "small" size case (65,536 (=64×32×32) meshes/node)

		T2K/Tokyo 512 nodes	**Cray XE6 128 nodes**	**Fujitsu FX10 128 nodes**
HB 4×4/6×4	Original	1.00	1.00	1.00
	AT	1.08	**<u>1.49</u>**	1.06
	Estimated Best	1.14	1.41	1.06
HB 8×2/12×2	Original	.963	.879	1.03
	AT	**<u>1.35</u>**	1.38	**<u>1.08</u>**
	Estimated Best	1.29	1.47	1.06
HB 16×1/24×1	Original	.572	.652	.866
	AT	1.12	1.14	.920
	Estimated Best	1.08	1.06	.932

5 Concluding Remarks

In the present work, we focus on automatic selection of single-threading or multi-threading in procedures of parallel multigrid solvers using hybrid parallel programming models. We propose a simple empirical method for AT of related parameters. The proposed method is based on the run-time tuning procedure for the optimization of communications as a subroutine of the target application and the parameter of the critical loop length for multi-threading derived from a simple off-line benchmark. The effect of the proposed method was evaluated on the T2K/Tokyo, the Cray XE6, and the Fujitsu FX10 using up to 8,192 cores. The proposed AT method is very effective, and the automatically tuned code provides twice the performance as the original code on the T2K/Tokyo and the Cray XE6 when the problem size per node is relatively small. The proposed method is very useful in strong scaling computations in these architectures. The effect is not so significant on the Fujitsu FX10, because multi-threading provides a higher efficiency even for short loops on the Fujitsu FX10 due to its hardware barrier. Because the original code is optimized for cc-NUMA architectures, such as the T2K/Tokyo and the Cray XE6, a different strategy for further optimization may be needed for the Fujitsu FX10. Generally speaking, NUMA-aware optimizations do not improve performance of the code on such architectures like Fujitsu FX10, where each core of the computing node can access the memory in a uniform manner.

In the present work, we focus on the choice of single-threading or multi-threading. A more sophisticated method that defines the optimum number of threads at each level may further contribute to optimization. For example, current choice is only 16-threads or a single thread for HB 16×1 of T2K/Tokyo and Fujitsu FX10, but 2-, 4- or 8-thread procedures at certain levels of the multigrid may provide further improvement of the performance. This type of more flexible approach is an interesting topic for future works. Because the topic of the present work covers only a small aspect of parallel multigrid methods, other directions of optimization, such as reducing communications, should also be considered in future works.

Acknowledgements. This work is supported by Core Research for Evolutional Science and Technology (CREST), Japan Science and Technology Agency (JST), Japan.

References

1. Nakajima, K.: Parallel Multigrid Solvers Using OpenMP/MPI Hybrid Programming Models on Multi-Core/Multi-Socket Clusters. In: Palma, J.M.L.M., Daydé, M., Marques, O., Lopes, J.C. (eds.) VECPAR 2010. LNCS, vol. 6449, pp. 185–199. Springer, Heidelberg (2011)
2. Nakajima, K.: New strategy for coarse grid solvers in parallel multigrid methods using OpenMP/MPI hybrid programming models. ACM Proceedings of the 2012 International Workshop on Programming Models and Applications for Multicores and Manycores, ACM Digital Library (2012), doi:10.1145/2141702.2141713
3. Information Technology Center, The University of Tokyo, `http://www.cc.u-tokyo.ac.jp/`
4. The T2K Open Supercomputer Alliance, `http://www.open-supercomputer.org/`
5. Baker, A., Gamblin, T., Schultz, M., Yang, U.: Challenge of Scaling Algebraic Multigrid across Modern Multicore Architectures. In: Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium (IPDPS 2011), pp. 275–286 (2011)
6. Hypre Library, `http://acts.nersc.gov/hypre/`
7. NERSC, Lawrence Berkeley National Laboratory, `http://www.nersc.gov/`
8. STREAM (Sustainable Memory Bandwidth in High Performance Computers), `http://www.cs.virginia.edu/stream/`
9. Deutsch, C.V., Journel, A.G.: GSLIB Geostatistical Software Library and User's Guide, 2nd edn. Oxford University Press (1998)
10. Washio, T., Maruyama, K., Osoda, T., Shimizu, F., Doi, S.: Efficient implementations of block sparse matrix operations on shared memory vector machines. In: Proceedings of the 4th International Conference on Supercomputing in Nuclear Applications (SNA 2000) (2000)