Large-scale Simulations of 3D Groundwater Flow using Parallel Geometric Multigrid Method

中 島 研 吾

東京大学情報基盤センター

本稿では，2012 年 12 月に実施された大規模 HPC チャレンジの結果を報告する。

本大規模 HPC チャレンジの成果をまとめた論文「Kengo Nakajima, Large-scale Simulations of 3D Groundwater Flow using Parallel Geometric Multigrid Method」は，2013 年 6 月に Barcelona で開催された ICCS 2013（International Conference on Computational Science）[1]の併設ワークショップ（IHPCES 2013 (Third International Workshop on Advances in High-Performance Computational Earth Sciences: Applications and Frameworks)）に採択され，Procedia Computer Science Vol.18（Elsevier）に掲載された。本稿は出版元である Elsevier 社の許可を受け次ページ以降に上記論文の全文を掲載するものである。

（日本語要旨）

OpenMP/MPI ハイブリッド並列プログラミングモデルに基づく多重格子法（multigrid）はポストペタ／エクサスケールのスーパーコンピュータシステムにおける大規模科学技術計算手法として注目されている。本研究では，係数行列の格納法が並列幾何学的多重格子法（parallel geometric multigrid）による線形方程式ソルバーの性能へ与える影響について評価し，Ellpack-ITpack（ELL）に基づく新しい並列データ構造が提案した。提案手法を三次元地下水流れシミュレーションコード pGW3D-FVM に適用し，計算性能と安定性を東京大学情報基盤センターFujitsu PRIMEHPC FX10（Oakleaf-FX）の最大 4,096 ノード，65,536 コアを使用して評価した。ELL 型係数格納法と Coarse Grid Aggregation（CGA）を組み合わせた並列多重格子ソルバーは従来の CRS 型係数格納法に基づくコードに対して Weak Scaling で 13%～35%，Strong Scaling で 40%～70%の性能改善が得られた。

[1] http://www.iccs-meeting.org/iccs2013/

Available online at www.sciencedirect.com
SciVerse ScienceDirect
Procedia Computer Science 18 (2013) 1265 – 1274
Procedia Computer Science

2013 International Conference on Computational Science

Large-scale Simulations of 3D Groundwater Flow using Parallel Geometric Multigrid Method

Kengo Nakajima[a],*

[a]*Information Technology Center, University of Tokyo, 2-11-16 Yayoi, Bunkyo-ku, Tokyo 113-8658, Japan*

Abstract

The multigrid method used with OpenMP/MPI hybrid parallel programming models is expected to play an important role in large-scale scientific computing on post-peta/exa-scale supercomputer systems. In the present work, the effect of sparse matrix storage formats on the performance of parallel geometric multigrid solvers was evaluated, and a new data structure for the *Ellpack-Itpack* (ELL) format is proposed. The proposed method is implemented for pGW3D-FVM, a parallel code for 3D groundwater flow simulations using the multigrid method, and the robustness and performance of the code was evaluated on up to 4,096 nodes (65,536 cores) of the Fujistu FX10 supercomputer system at the University of Tokyo. The parallel multigrid solver using the ELL format with coarse grid aggregation provided excellent performance improvement in both weak scaling (13%–35%) and strong scaling (40%–70%) compared to the original code using the CRS format.

Keywords: Groudwater Simulations; Multigrid; Preconditioning; OpenMP/MPI Hybrid Parallel Programming Model

1. Introduction

To achieve minimal parallelization overheads on multi-core clusters, a multi-level hybrid parallel programming model is often employed. With this model, 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. In previous works [1,2], OpenMP/MPI hybrid parallel programming models were implemented in *pGW3D-FVM*, a 3D finite-volume simulation code for groundwater flow problems through heterogeneous porous media, by using parallel iterative solvers with multigrid preconditioning. The performance and the robustness of the

* Corresponding author. Tel.: +81-3-5841-2719; fax: +81-3-5841-2708 .
E-mail address: nakajima@cc.u-tokyo.ac.jp .

1877-0509 © 2013 The Authors. Published by Elsevier B.V.
Selection and peer review under responsibility of the organizers of the 2013 International Conference on Computational Science
doi:10.1016/j.procs.2013.05.293

developed code were evaluated on the T2K Open Supercomputer (T2K/Tokyo) and the Fujitsu FX10 System (Oakleaf-FX) at the University of Tokyo [3] by using up to 4,096 nodes (65,536 cores) for both weak and strong scaling computations. A new strategy for solving equations at the coarsest level (coarse grid solver) is proposed and evaluated in this paper. The *coarse grid aggregation* (CGA) proposed in [2] improves the performance and the robustness of multigrid procedures with large numbers of MPI processes. Generally, OpenMP/MPI hybrid parallel programming models with a larger number of threads per MPI process provide excellent performance for both weak and strong scaling computations with a large number of computing nodes. The concepts of OpenMP/MPI hybrid parallel programming models can be easily extended and applied to supercomputers with heterogeneous computing nodes with accelerators/co-processors, such as GPUs and/or many-core processors (Intel Many Integrated Core Architecture). A multigrid is a scalable method for solving linear equations and preconditioning Krylov iterative linear solvers, and 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.

In the present work, the effect of sparse matrix storage formats was evaluated. In previous work [1,2], the *compressed row storage* (CRS) format was used. In this previous work, the *Ellpack-Itpack* (ELL) format was also evaluated. Although ELL provides much better performance than does CRS, it is not suitable for ILU/IC-type preconditioning with reordering on massively parallel computers. A new data structure for the ELL format is proposed here and evaluated on up to 4,096 nodes (65,536 cores) of the Fujitsu FX10 [3]. Generally, the performance of solvers with ELL is much better than those with CRS, especially if the number of threads for each MPI process is smaller. The rest of this paper is organized as follows. In Section 2, an overview of the target hardware and application is provided. In Section 3, we give a summary of the new data structure for the ELL format. Finally, Section 4 provides the results of computations, and related work and final remarks are offered in Sections 5 and 6.

2. Hardware Environment and Target Application

2.1. Hardware Environment

The Fujitsu FX10 system at the University of Tokyo (Oakleaf-FX) [3] is Fujitsu's PRIMEHPC FX10 massively parallel supercomputer with a peak performance of 1.13 PFLOPS. Fujitsu FX10 consists of 4,800 computing nodes of SPARC64™ IXfx processors with sixteen cores (1.848 GHz) (Fig. 1) [4]. The SPARC64™ IXfx incorporates many features for high-performance computing (HPC), which includes the SPARC64™ V9 specification with arithmetic computational extensions (HPC-ACE) and a hardware barrier for high-speed synchronization of on-chip cores [4]. The entire system consists of 76,800 cores and 154 TB memory. Each core has a 64 KB L1 instruction/data cache. A 12 MB L2 cache is shared by sixteen cores on each node. On the SPARC64™ IXfx, each of the sixteen cores accesses memory in a uniform manner. Nodes are connected via a 6-dimensional mesh/torus interconnect, called *Tofu*. In the present work, up to 4,096 nodes of the system were evaluated. Although users can specify the topology of the network on Fujitsu FX10, this capability was not used in the current study.

Fig. 1. Fujitsu's SPARC64^{TM} IXfx Processor [4].

2.2. pGW3D-FVM

pGW3D-FVM, a parallel simulation code based on the finite-volume method (FVM), solves groundwater flow problems through saturated heterogeneous porous media (Fig. 2). The problem is described by the following Poisson equation and boundary condition (1):

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

where ϕ denotes the potential of the water head, λ(x,y,z) describes the water conductivity, and q is the value of the volumetric flux of each finite-volume mesh and is set to a uniform value (=1.0) in this work. A heterogeneous distribution of water conductivity in each mesh is calculated by a sequential Gauss algorithm, which is widely used in the area of geostatistics [5]. 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 mesh is a cube, and distribution of the meshes is structured as finite-difference-type voxels. In this work, an entire model consists of clusters of small models with 128^3 meshes. In each small model, the distribution pattern is the same. Therefore, the same pattern of heterogeneity appears periodically. pGW3D-FVM is parallelized by domain decomposition using MPI for the communications between partitioned domains [1,2].

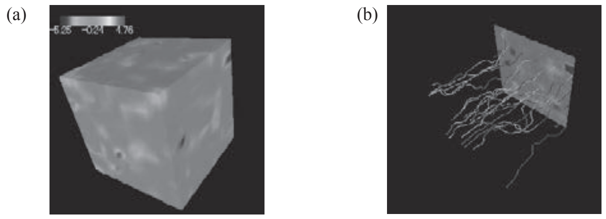

Fig. 2. Example of groundwater flow through heterogeneous porous media. (a) Distribution of water conductivity; (b) Streamlines.

2.3. Iterative Method with Multigrid Preconditioning

The conjugate gradient (CG) solver with multigrid preconditioner (MGCG) [1,2] was applied for solving Poisson's equations with symmetric positive definite (SPD) coefficient matrices derived by the pGW3D-FVM code. Iterations were repeated until the norm |r|/|b| was less than 10^{-12}. The multigrid method is an example of a scalable linear solver and is widely used for large-scale scientific applications. Relaxation methods such as the Gauss-Seidel method can efficiently damp high-frequency error, but low-frequency error remains. The multigrid idea is to recognize that this low-frequency error can be accurately and efficiently solved on a coarser grid [6]. A very simple geometric multigrid with a *V-cycle* algorithm is applied, where 8 children form 1 parent mesh in an isotropic manner for structured finite-difference-type voxels. The *level* of the finest grid is set to 1 and is numbered from the finest to the coarsest grid, at which the number of meshes is 1 at each domain (MPI process). Multigrid operations at each level are done in a parallel manner, but the operations at the coarsest level are executed on a single core by gathering the information of entire processes. The total number of meshes at the coarsest level is equal to the number of domains (MPI processes). In the multigrid procedure, equations at each level are *relaxed* by smoothing operators, such as Gauss-Seidel iterative solvers. Many types of smoothing operators have been proposed and used [6]. Among those, Incomplete LU/Cholesky factorization without fill-ins (ILU(0)/IC(0)) are widely used. These smoothing operators demonstrate excellent robustness for ill-conditioned problems [1,2]. In this study, IC(0) is adopted as a smoothing operator. The IC(0) process includes global operations and it is difficult to be parallelized. The block-Jacobi-type localized procedure is an option for distributed parallel operations, but this approach tends to be unstable for ill-conditioned problems. To stabilize the localized IC(0) smoothing, the *additive Schwarz domain decomposition* (ASDD) method for overlapped regions [7] is introduced.

2.4. Procedures for Reordering

The pGW3D-FVM 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 cells 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. Reverse Cuthill-McKee (RCM) reordering facilitates faster convergence of iterative solvers with ILU/IC preconditioners than does 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) [8]. In this method, further renumbering in a cyclic manner is applied to the vertices reordered by RCM. In CM-RCM, the number of colors should be large enough to ensure that vertices of the same color are independent.

2.5. CGA

In [1], V-cycle multigrid processes were applied as shown in Fig. 3. In previous work [2], CGA was proposed, where operations for *aggregation/disaggregation of MPI processes* at 2) and 4) in Fig. 3 are done at a finer level. If we switch to a *coarse grid solver* at a finer level, more robust convergence and reduction of communication overhead are expected, even though the size of the *coarse grid problem* is larger than that of the original configuration. Furthermore, the coarse grid solver is multi-threaded by OpenMP and uses all cores on each MPI process, although only a single core on each node was utilized in [1]. In post-peta/exa-scale systems, each node may consist of O(10^2) cores. Therefore, utilization of these *many cores* on each node should be considered. Figure 4 shows an example of CGA. In this case, information of each MPI process is gathered in a single MPI process after computation at *level=2*. Thus, the stage of the coarse grid solver starts earlier than does the original case. CGA is not applied to *flat MPI*.

1) Starting from finest level (level=1), a smoothing operation by IC(0), and *restriction* to coarser levels are applied. Operations at each MPI process are done by a parallel manner with some communication.
2) At the coarsest level, information on each process is gathered into a single MPI process, and a “coarse grid problem” is formed. The size of this coarse grid problem corresponds to the number of MPI processes.
3) **(Coarse Grid Solver):** A serial multigrid solver by IC(0) is applied for solving the coarse grid problem on a single core.
4) After the coarse grid problem is solved, the results of the coarse grid solver are scattered to each MPI process.
5) Starting from the coarsest level, *prolongation* to a finer level with smoothing at each level, is applied, until the finest level (level=1).

Fig. 3. Parallel multigrid method with V-cycle [1]

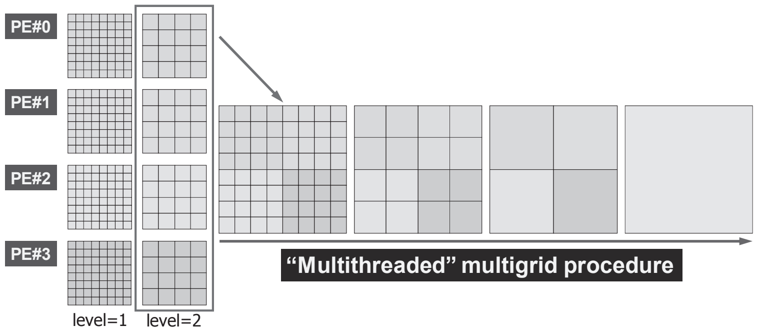

Fig. 4. Example of *coarse grid aggregation* (CGA), where information of each MPI process is gathered in a single MPI process after computation at *level=2*.

3. Sparse Matrix Storage Formats

3.1. CRS and ELL

Generally, computations with sparse matrices are *memory-bound* processes, because of the *indirect* memory accesses. Various types of storage formats have been proposed [9]. The c*ompressed row storage* (CRS) format is the most popular and widely used because of its flexibility. It stores only non-zero components of sparse matrices, as shown in Fig. 5(a). In the previous work [1,2], the CRS format was used. In the *Ellpack-Itpack* (ELL) format, the number of non-zero components of each row is set to that of the longest non-zero entry row of the matrix, as shown in Fig. 5(b). This format allows one to achieve better performance for memory access than CRS, but introduces extra computations and memory requirements, since some rows are zero-padded, as shown in Fig. 5(b). Generally, ELL is suitable for coefficient matrices derived from simple structured meshes, where the number of non-zero components at each row is almost fixed.

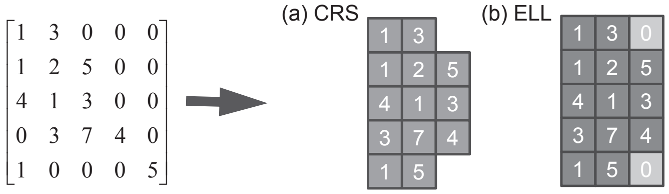

Fig. 5. Sparse matrix storage formats. (a) Compressed row storage (CRS); (b) Ellpack-Itpack (ELL).

3.2. Implementation of ELL with New Data Structure

Although structured meshes are used in pGW3D-FVM, utilization of ELL is not straightforward. In pGW3D-FVM, each of the *diagonal*, *lower triangular*, and *upper triangular* components of the coefficient matrices are separately stored in different arrays for IC(0) smoothing operations. If reordering by coloring is applied, the number of lower/upper triangular components of each row may change. Moreover, the distributed local data structure of pGW3D-FVM may require significant extra computations by zero-padded components, as shown in Fig. 5(b). In Fig. 6, (a) and (b) provide the local data structure of pGW3D-FVM. The numbering of *external* cells on each distributed mesh starts after all *internal* cells in Fig. 6(a) have been numbered. In the structured mesh in pGW3D-FVM, the initial numbering is *lexicographical*. Therefore, each *pure internal cell* has three lower and three upper triangular components, as shown in Fig. 6(b). But, an *internal cell on a domain boundary* may have more than three upper triangular components, as shown in Fig. 6(b), because the IDs of the external cells are larger than those of the internal cells. The maximum number of upper triangular components may be six at internal cells on the domain boundary.

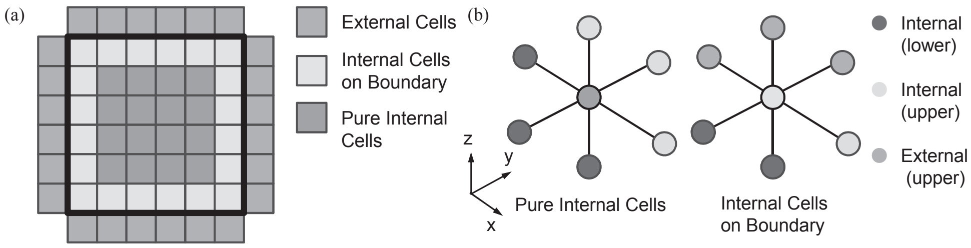

Fig. 6. Local data structure of pGW3D-FVM. (a) Internal/external cells; (b) Upper/lower components of internal cells.

In the present work, the ELL format is applied to pGW3D-FVM. Cuthill-McKee (CMK) reordering is used, because CMK does not change the inequality relationship of the ID of each cell for the structured meshes used in pGW3D-FVM. Convergence of ILU/IC preconditioning by CMK is slightly worse than that of RCM. Moreover, at each level of CMK, pure internal cells and internal cells on the boundary are calculated separately. Because reordering on a same level (or color) does not affect convergence [8], computations related to upper triangular components such as the backward substitution process in IC(0) smoothing for internal cells on the boundary (6 upper components) are done first, then pure internal cells (3 upper components) are calculated.

4. Results

4.1. Overview

The performance of the developed code was evaluated on up to 4,096 (65,536 cores) of the Fujitsu FX10. IC(0) smoothing was applied twice at each level, and a single cycle of ASDD was conducted at each smoothing operation. A single V-cycle operation was applied as a preconditioning process of each CG iteration. At the coarsest level, IC(0) smoothing was applied once. The following three types of OpenMP/MPI hybrid parallel programming models were applied, and the results were compared with those of *flat MPI*:

- **Hybrid 4×4 (HB 4×4)**: Four OpenMP threads for each MPI process, four MPI processes on each node
- **Hybrid 8×2 (HB 8×2)**: Eight OpenMP threads for each MPI process, two MPI processes on each node
- **Hybrid 16×1 (HB 16×1)**: Sixteen OpenMP threads for each MPI process, a single MPI process on each node.

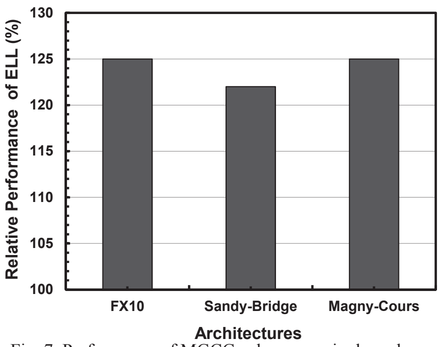

Fig. 7. Performance of MGCG solver on a single node, relative performance of MGCG solver with ELL compared to that with CRS, Problem size/core= 262,144 (=64^3), based on elapsed time.

Moreover, two types of sparse matrix storage formats (CRS and ELL) were evaluated. CM-RCM reordering was applied to CRS, whereas CMK was used in ELL.

4.2. Comparison of CRS and ELL

First, the effect of sparse matrix storage formats was on a single node of several computer systems, including Fujitsu FX10. RCM reordering was applied to CRS. The number of iterations until convergence for CRS with RCM and that of ELL with CMK were the same. Fig.7 shows the relative performance of the MGCG solver with ELL compared to that of CRS on a single node of the Fujitsu FX10, Intel Sandy-Bridge and the AMD Opteron (Magny-Cours), where the problem size for each core is fixed as 262,144 (=64^3). The MGCG solver with ELL is 1.22–1.25 times as fast as that with CRS, although ELL needed extra computations. Table 1 shows the results of the performance analyzer for Fujitsu FX10 [3]. Number of instructions is reduced by ELL.

Next, the effect of reordering for CRS cases was evaluated using 4 nodes (64 cores) of Fujitsu FX10 for *flat*

Table 1. Performance of MGCG solver for 262,144 cells (=64^3) with a single core of Fujitsu FX10 by a performance analyzer for FX10 [4].

	Memory Access Throughput	Instructions	Elapsed Time	Operation Wait
CRS	76.9% of peak	1.447×10^{10}	6.815 sec.	1.453 sec.
ELL	75.3% of peak	6.385×10^{9}	5.457 sec.	0.312 sec.

MPI and OpenMP/MPI hybrid parallel programming models. The number of finite-volume meshes per core was 262,144 (=64^3), and therefore the total problem size was 16,777,216. In Fig. 8, (a) and (b) show the relationship between performance (computation time for linear solvers) and number of colors for each parallel programming model. CGA was not applied. Generally speaking, CM-RCM with 2 or 4 colors (CM-RCM(2) or CM-RCM(4)) provides the best performance of computation time per iteration, as shown in Fig. 8(a). This is because that cache is the most efficiently utilized in CM-RCM(2) for the structured finite-difference-type voxels used in pGW3D-FVM [1,2]. But, the number of iterations for convergence decreases as the number of colors increases, as shown in [1,2]. The optimum number of colors according to the elapsed computation time for CRS is 190 (flat MPI), 70 (HB 4×4), 75 (HB 8×2), and 100 (HB 16×1), as shown in Fig. 8(b). The difference between the performances of CRS and ELL is not so significant as cases with a single core, shown in Fig. 7, because of memory contention. The performance of MGCG with ELL decreases as the number of threads for each MPI process increases. Because ELL adopts CMK reordering, the number of colors is relatively larger than the optimum cases of CRS. Therefore, the overhead of OpenMP is more significant, if the number of threads for each MPI process increases. Finally, HB 16×1 with CRS (CM-RCM(100)) and HB 16×1 with ELL are almost competitive; the performance of ELL is 15%–27%, which is better than that of CRS in other parallel programming models.

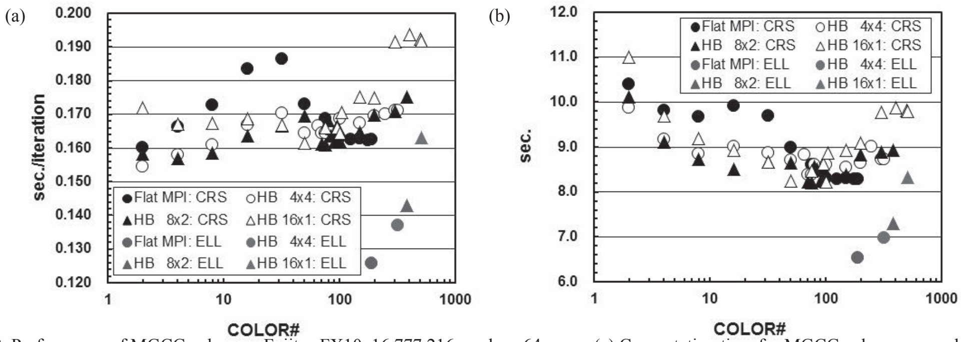

Fig. 8. Performance of MGCG solver on Fujitsu FX10, 16,777,216 meshes, 64 cores. (a) Computation time for MGCG solvers per each iteration; (b) Elapsed computation time for MGCG solvers without *coarse grid aggregation* (CGA).

4.3. Weak Scaling

The performance of weak scaling was evaluated by using 8 nodes (128 cores) and 4,096 nodes (65,536 cores) of the Fujitsu FX10. The number of finite-volume meshes per core was 262,144 (=64^3); therefore, the maximum total problem size was 17,179,869,184 meshes. In the CRS cases, the number of colors for CM-RCM reordering was set to 190 (flat MPI), 70 (HB 4×4), 75 (HB 8×2), and 100 (HB 16×1), in accordance with the results stated above. The performance of solvers using CRS, ELL, and ELL with CGA (ELL-CGA) were evaluated. In Fig. 9, (a) and (b) compare the performances of CRS and ELL on weak scaling using 4,096 nodes (65,536 cores) for HB 4×4 and HB 8×2 for evaluating the effect of switching level for a coarse grid solver in ELL-CGA. A 2-digit number in each label after a colon on the transverse axis of Fig.9 (a) and (b) indicates the number of iterations until convergence. In CRS and ELL, *localized* IC(0) smoothing with ASDD was applied to each MPI process if the *level* was smaller than the coarsest level. Because the coarse grid solver was performed on a single MPI process in ELL-CGA, the convergence provided by IC(0) smoothing is much more robust than that by *localized* IC(0) smoothing [2]. Although convergence is significantly improved by switching to a coarse grid solver at a finer level (*i.e.*, smaller number of *level*), the problem size of the coarse grid solver is larger for switching at *level*=6, and the computation is rather expensive, as shown in Fig. 9. The optimum switching level is "*level*=7" for both HB 4×4 and HB 8×2. Moreover, switching at "*level*=7" also provides the optimum solution for HB 16×1.

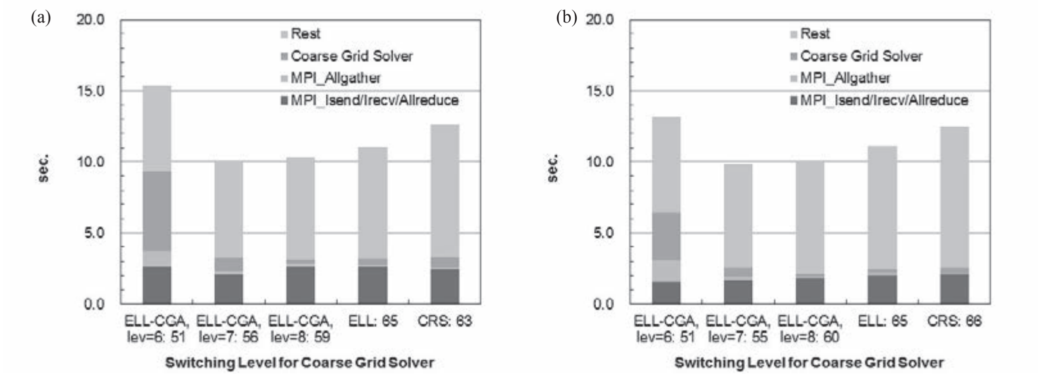

Fig. 9. Performance of MGCG solver on Fujitsu FX10 using 4,096 nodes (65,536 cores), total problem size: 17,179,869,184 meshes, comparison of CRS and ELL, effect of *switching level* for coarse grid solver in ELL-CGA. (a) HB 4×4; (b) HB 8×2.

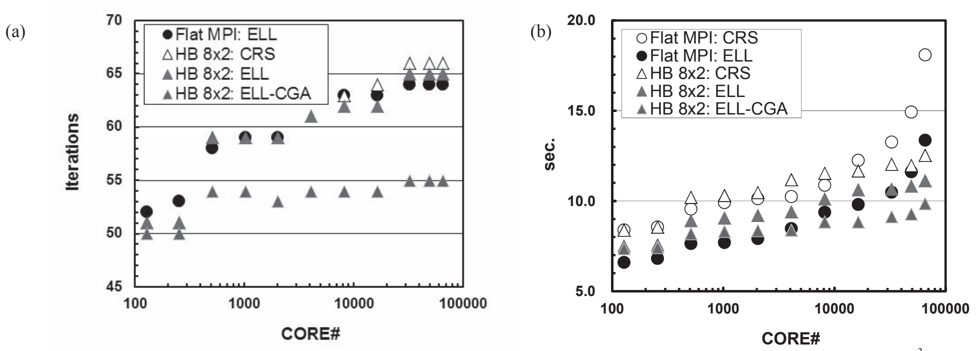

Fig. 10. Performance of MGCG solver on Fujitsu FX10 using up to 4,096 nodes (65,536 cores), weak scaling: 262,144 (=64^3) meshes/core, max. total problem size: 17,179,869,184 meshes. (a) Number of iterations for MGCG solvers until convergence; (b) Elapsed computation time for MGCG solvers, CM-RCM (190): flat MPI with ELL (75): HB 8×2 with ELL, switching at *level*=7 for HB 8×2 with ELL-CGA.

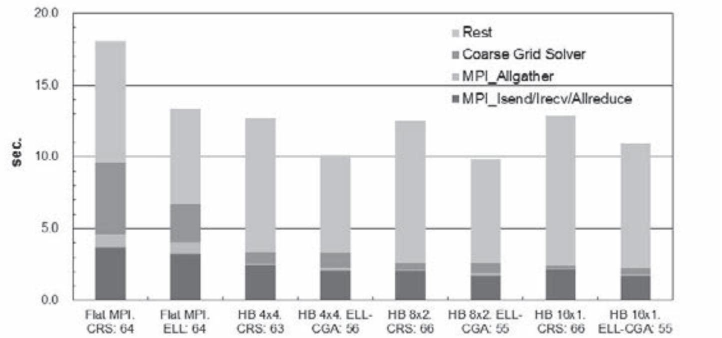

Fig. 11. Performance of MGCG solver on Fujitsu FX10 using 4,096 nodes (65,536 cores), total problem size: 17,179,869,184 meshes, switching at *level*=7 for ELL-CGA cases.

Figures 10 and 11 provide the results of weak scaling. Improvement of the performance from CRS to ELL-CGA is 13%–35% at 4,096 nodes. Switching at "*level*=7" was applied to all ELL-CGA cases. HB 8×2 with ELL-CGA provides the best performance at 4,096 nodes. Figure 11 shows that the ratio of the *coarse grid solver* is very large at 4,096 nodes in a *flat MPI*. This is because the initial problem size for the coarse grid solver is larger, and coarse grid problems are solved by a single core in *flat MPI* cases. In contrast, the size of the *coarse grid problem* is small and solved by 16 threads in HB 16×1.

4.4. Strong Scaling

Finally, the performance of strong scaling was evaluated for a fixed size of problem with 268,435,456 meshes (=1024×512×512) using 8 nodes (128 cores) to 4,096 nodes (65,536 cores) of Fujitsu FX10. At 4,096 nodes, the problem size per each core is only 4,096 meshes (=16^3). CM-RCM(16) was applied to the CRS cases. Figure 12 shows the performance of the MGCG solver. The performance of *flat MPI* with 8 nodes (128 cores) is 100%, and the parallel performance is 40%–50% up to 512 nodes (8,192 cores). HB 16×1 with ELL-CGA provides slightly better performance than do the others at 4,096 nodes. Figure 13 and Table 2 show the performance at 4,096 nodes. ELL-CGA for the three hybrid parallel programming models improved the performance by 20% for the number of iterations until convergence, and by 40%–70% for the elapsed computation time. This effect is significant for reducing the time of the coarse grid solver.

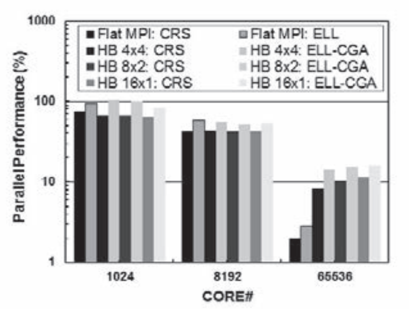

Fig. 12. Performance of MGCG solver on Fujitsu FX10 using up to 4,096 nodes (65,536 cores), strong scaling: 268,435,456 meshes (=1024×512×512), parallel performance based on the performance of *flat MPI* with 8 nodes (128 cores), CM-RCM(16) for CRS cases.

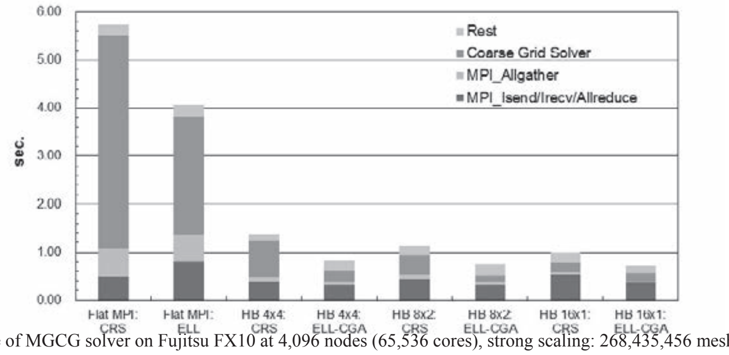

Fig. 13. Performance of MGCG solver on Fujitsu FX10 at 4,096 nodes (65,536 cores), strong scaling: 268,435,456 meshes (=1024×512×512), CM-RCM(16) for CRS cases.

Table 2. Parallel performance of MGCG solver, strong scaling: 268,435,456 meshes, 4,096 nodes (65,536 cores), parallel performance based on the performance of *flat MPI* with 8 nodes (128 cores).

	Flat MPI		HB 4×4		HB 8×2		HB 16×1	
	CRS	ELL	CRS	ELL-CGA	CRS	ELL-CGA	CRS	ELL-CGA
Iterations until Convergence	57	58	58	46	63	49	63	51
MGCG solver (sec.)	5.73	4.07	1.38	.816	1.13	.749	1.00	.714
Parallel performance (%)	2.02	2.85	8.38	14.2	10.3	15.5	11.6	16.2

5. Related Work

In [10], the performance of parallel programming models for algebraic multigrid solvers in the *Hypre* library [11] was evaluated on various multicore HPC platforms with more than 10^5 cores. The 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 on the cc-NUMA architecture.

6. Summary and Future Work

The effect of a sparse matrix storage format on the performance of parallel MGCG solvers with the OpenMP/MPI hybrid parallel programming model was evaluated on up to 4,096 nodes of the Fujistu FX10 at the University of Tokyo. A new data structure for the ELL format was proposed and implemented for pGW3D-FVM. The ELL format provided excellent improvement of memory access throughput, and the parallel multigrid solver using the ELL format with CGA provided excellent performance and robustness. The effect of the ELL format on performance was more significant for parallel programming models with a fewer number of threads on each MPI process, such as *flat MPI* and HB 4×4. The performance improvement from the original solver with CRS to the new one with ELL-CGA at 4,096 nodes was 13%–35% for weak scaling, and 40%–70% for strong scaling. We need to investigate further optimization for efficient communication of the code. Although the optimum switching level for CGA was derived through parameter studies, a method for automatic selection of the level is an interesting technical issue for future work. Development of the CGA by using multiple MPI processes is another technical issue for large numbers of processes on future architectures with smaller amounts of memory for each node.

Acknowledgements

This work is supported by Core Research for Evolutional Science and Technology (CREST), the Japan Science and Technology Agency (JST), Japan. The computational resource of Fujitsu FX10 was awarded by the "Large-scale HPC Challenge" Project, Information Technology Center, the University of Tokyo.

References

[1] Nakajima, K., 2012. "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 & Applications for Multi/Manycores

[2] Nakajima, K., OpenMP/MPI Hybrid Parallel Multigrid Method on Fujitsu FX10 Supercomputer System, IEEE Proceedings of 2012 International Conference on Cluster Computing Workshops, IEEE Digital Library: 10.1109/ClusterW.2012.35; 2012, p.199-206

[3] Information Technology Center, The University of Tokyo: http://www.cc.u-tokyo.ac.jp/

[4] Fujitsu: http://www.fujitsu.com/

[5] Deutsch, C.V., Journel, A.G. *GSLIB Geostatistical Software Library and User's Guide, Second Edition*. Oxford University Press; 1998

[6] Tottemberg, U., Oosterlee, C., Schuller, A. *Multigrid*, Academic Press; 2001

[7] Smith, B., Bjϕrstad, P., Gropp, W. *Domain Decomposition, Parallel Multilevel Methods for Elliptic Partial Differential Equations*, Cambridge Press; 1996

[8] Washio, T., Maruyama, K., Osoda, T., Shimizu, F., Doi, S., 2000. "Efficient implementations of block sparse matrix operations on shared memory vector machines", Proceedings of The 4th International Conference on Supercomputing in Nuclear Applications

[9] Saad, Y. *Iterative Methods for Sparse Linear Systems Second Edition*, SIAM; 2003

[10] Baker, A., Gamblin, T., Schultz, M., Yang, U., Challenge of Scaling Algebraic Multigrid across Modern Multicore Architectures, Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium (IPDPS'11); 2011, p.275-286

[11] Hypre Library: http://acts.nersc.gov/hypre/