Optimization of Serial and Parallel Communications for Parallel Geometric Multigrid Method

中島研吾

東京大学情報基盤センター

本稿では，2013 年 11 月，2014 年 5 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, Optimization of Serial and Parallel Communications for Parallel Geometric Multigrid Method」は，2014 年 12 月に新竹（台湾）で開催された IEEE ICPADS 2014（20th International Conference for Parallel and Distributed Systems）[1]に採択され，Proceedings（IEEE Digital Library）に掲載されるとともに，Best Paper Award を受賞した。ICPADS 2014 は合計 322 件の投稿があり，そのうち 96 件が採択され（採択率 29.8%）更に，そのうち 4 件が Best Paper Candidates として選出され，最終的に本論文が Best Paper Award を受賞した。本稿は出版元である IEEE 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。

[1] http://www.icpads.org/

Optimization of Serial and Parallel Communications for Parallel Geometric Multigrid Method

Kengo Nakajima

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

***Abstract*— The parallel multigrid method is expected to play an important role in large-scale scientific computing on post-peta/exa-scale supercomputer systems, and it also includes serial and parallel communication processes which are generally expensive. In the present work, new format for sparse matrix storage based on *sliced* Ellpack-Itpack (ELL) format is proposed for optimization of serial communication in data transfer through memories, and hierarchical coarse grid aggregation (*h*CGA) is introduced for optimization of parallel communication by message passing. The proposed methods are 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 *sliced* ELL format provided performance improvement in both weak scaling (25%–31%) and strong scaling (9%–22%) compared to the code using the original ELL format. Moreover, *h*CGA provided excellent performance improvement in both weak scaling (1.61 times) and strong scaling (6.27 times) for *flat MPI* parallel programming model.**

Keywords—parallel computing; iterative solvers; multigrid; communication; matrix storage format; multicore

I. Introduction

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 *parallel* multigrid method is expected to be one of the powerful tools on post-peta/exa-scale systems. Recently, HPCG (High Performance Conjugate Gradients) [1] was proposed as a new benchmark for evaluation of the practical performance of supercomputer systems. HPCG solves sparse matrices derived from finite-element application using conjugate gradient linear solver (CG) preconditioned with multigrid method.

In previous works by the author [2,3,4], OpenMP/MPI hybrid parallel programming models were implemented for *pGW3D-FVM*, a 3D finite-volume simulation code for groundwater flow problems through heterogeneous porous media, by using parallel conjugate gradient (CG) solver with multigrid preconditioner (MGCG). The performance and the robustness of the developed code were evaluated on multicore clusters, such as the T2K Open Supercomputer (T2K/Tokyo) and the Fujitsu FX10 System (Oakleaf-FX) at the University of Tokyo [5], by using up to 4,096 nodes (65,536 cores) for both weak and strong scaling computations.

It is well-known that convergence of the solver at the coarsest level of the multigrid cycle (*coarse grid solver*) strongly affects convergence of the entire process of multigrid [2,6,7,8]. The *coarse grid aggregation* (CGA) proposed in [3] improves the performance and the robustness of multigrid procedures with large numbers of MPI processes. In [4], the effect of a format of sparse matrix storage on the performance of MGCG was evaluated. The *Ellpack-Itpack* (*ELL*) format was applied to pGW3D-FVM, and it provided excellent improvement of memory access throughput, and the MGCG solver using the ELL format with CGA showed excellent scalable performance and robustness. The performance improvement from the original solver with the *compressed row storage* (CRS) format in [3] to the new one with ELL-CGA at 4,096 nodes of the Fujitsu FX10 System (Oakleaf-FX) was 13%–35% for weak scaling, and 40%–70% for strong scaling.

The parallel multigrid method and MGCG include both of serial and parallel communication processes which are generally expensive. The *serial* communication is the data transfers through memory hierarchies of each processor. Because MGCG solver with sparse coefficient matrices is a memory-bound process, serial communication is a serious problem. In the present work, further modification is introduced into the ELL format of the coefficient matrices. The parallel communication is by message passing between computing nodes through the network using MPI. Hierarchical coarse grid aggregation (*h*CGA) is introduced for optimization of the parallel communication in the present work.

The rest of this paper is organized as follows. In Section II, an overview of the target hardware and application is provided, and results of the previous works are briefly overviewed. In Section III, we give a summary of the method for optimization of serial and parallel communications. Finally, Section IV provides the results of computations, and final remarks are offered in Sections V.

II. Hardware Environment and Target Application

A. Hardware Environment

The Fujitsu FX10 system at the University of Tokyo (Oakleaf-FX) [9] is Fujitsu's PRIMEHPC FX10 massively parallel supercomputer with a peak performance of 1.13 PFLOPS. The Oakleaf-FX system consists of 4,800 computing nodes of SPARC64™ IXfx with 16 cores (1.848 GHz) [9]. 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 16 cores on each node. On the SPARC64™ IXfx, each of the 16 cores accesses memory in a uniform manner. Nodes are connected via a 6-dimensional mesh/torus interconnect, called *Tofu*. Although users can specify the topology of the network on Fujitsu FX10, this capability was not used in the present work.

B. pGW3D-FVM with MGCG Solver

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

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

where ϕ denotes the potential of the water head, $\lambda(x,y,z)$ 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.

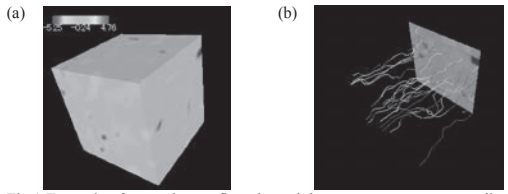

Fig.1 Example of groundwater flow through heterogeneous porous media. (a) Distribution of water conductivity; (b) Streamlines

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 [10]. The minimum and maximum values of water conductivity are 10^{-5} and 10^{5}, respectively, with an average value of 1.0. Each mesh is a cube, and distribution of the meshes is structured as finite-difference-type voxels. In the present work, an entire model consists of clusters of small models with 128^3 meshes with same pattern [2,3,4]. The conjugate gradient (CG) solver with multigrid preconditioner (MGCG) [2,3,4] was applied for solving Poisson's equations with symmetric positive definite (SPD) coefficient matrices derived by pGW3D-FVM. 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 the *level* is numbered from the finest to the *coarsest grid*, at which the number of meshes is 1 at each MPI process. Incomplete Cholesky factorization without fill-ins (IC(0)) is adopted as a smoothing operator of multigrid process for ill-conditioned problems. The *additive Schwarz domain decomposition* (ASDD) for overlapped regions [11] is introduced for stabilization of the block-Jacobi-type localized procedure of parallel IC(0). The pGW3D-FVM code is parallelized by domain decomposition using MPI [2,3,4]. In the OpenMP/MPI hybrid parallel programming model, multithreading by OpenMP is applied to each partitioned domain. The reordering of meshes in each domain allows the construction of local operations without global dependency for achieving parallel IC operations in multigrid processes. Cuthill-McKee (CMK), Reverse Cuthill-McKee (RCM), and RCM with cyclic multicoloring (CM-RCM) [2,3,4] are implemented.

C. CGA

In [2], multigrid processes by V-cycle were applied as shown in Fig. 2. Multigrid operations at each level are done in a parallel manner, but the operations at the coarsest level (*coarse grid solver*) 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 MPI processes. Moreover, overhead of parallel communication is significant at coarser level of the V-cycle.

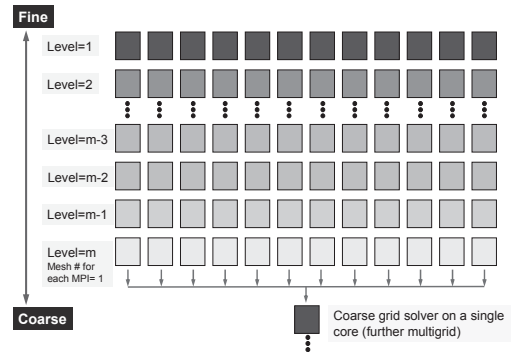

Fig.2. Original parallel multigrid method with V-cycle (restriction) [2]

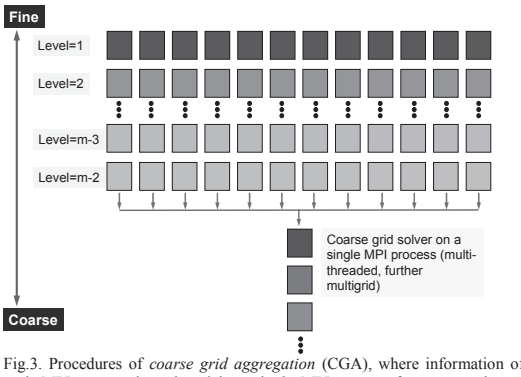

Fig.3. Procedures of *coarse grid aggregation* (CGA), where information of each MPI process is gathered in a single MPI process for computation at *level=m-2*

In [3], CGA was proposed, where operations for *aggregation/disaggregation of MPI processes* in Fig. 2 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 the original method [2]. Figure 3 shows procedures of CGA, where information of each MPI process is gathered in a single MPI process for computation at *level=m-2*. Thus, the stage of the coarse grid solver starts earlier than that starts in the original case. CGA is not applied to *flat MPI*.

D. CRS and ELL

Generally, computations with sparse matrices, such as multigrid and MGCG, are *memory-bound* processes, because of the *indirect* memory accesses. Various types of storage formats have been proposed. 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. 4(a). 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. 4(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. 4(b).

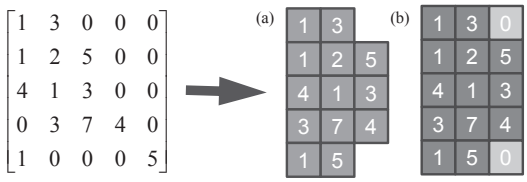

Fig. 4. Formats of sparse matrix storage. (a) Compressed row storage (CRS); (b) Ellpack-Itpack (ELL)

E. Results in the Previous Works

The performance of the developed code was evaluated on 8 to 4,096 nodes of the Fujitsu FX10. The following three types of OpenMP/MPI hybrid parallel programming tablemodels 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.

The number of finite-volume meshes per core was 262,144 (=64^3). The performance of solvers using CRS, ELL, and ELL with CGA (ELL-CGA) were evaluated. Figure 5 compares the performances of HB 8×2 at 4,096 nodes (65,536 cores). A 2-digit number in each label after a colon on the transverse axis of Fig.5 indicates the number of iterations until convergence. 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 [3].

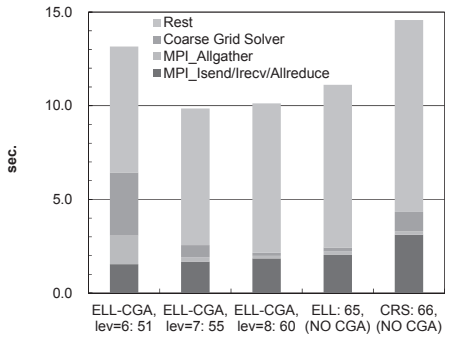

Fig. 5. Performance of MGCG solver on Fujitsu FX10 using 4,096 nodes (65,536 cores), total problem size: 17,179,869,184 meshes, comparison of CRS and ELL, effect of *switching level* for coarse grid solver in ELL-CGA, HB 8×2

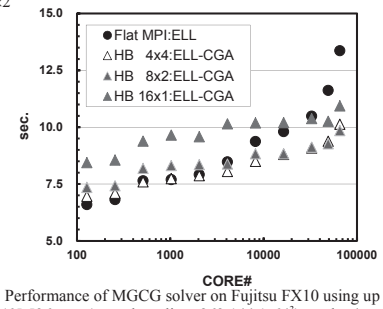

Fig. 6. Performance of MGCG solver on Fujitsu FX10 using up to 4,096 nodes (65,536 cores), weak scaling: 262,144 (=64^3) meshes/core, max. total problem size: 17,179,869,184 meshes, elapsed computation time for MGCG, switching to coarse grid solver at *level*=7 for ELL-CGA cases

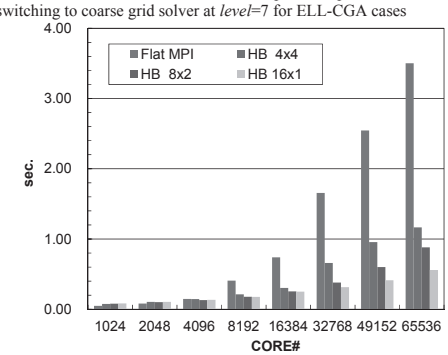

Fig. 7. Computation time for the coarse grid solver of MGCG solver on Fujitsu FX10 using 4,096 nodes (65,536 cores), total problem size: 17,179,869,184 meshes for ELL-CGA cases

The optimum parameter is "*level*=7" in this case, which has been determined through empirical parameter studies. Improvement of the performance from CRS to ELL-CGA is 13%–35% at 4,096 nodes [3]. Figure 6 provides the results of weak scaling. Switching at "*level*=7" was applied to all ELL-CGA cases. Although each of HB 4×4, HB 8×2 and HB 16×1 provides excellent scalability, HB 8×2 with ELL-CGA

provides the best performance at 4,096 nodes. *Flat MPI* with more than 10^4 cores cannot achieve scalability.

Figure 7 shows that the computation time spent for the *coarse grid solver* is getting larger for *flat MPI* with more than 10^4 cores. 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* for HB 16×1 is smaller and the problem is solved by 16 threads in HB 16×1. Computational amount of coarse grid solver for each core of *flat MPI* is 256 (=16×16) times as large as that of HB 16×1.

III. Optimization of Serial and Parallel Communications

F. Sliced ELL for Serial Communication

In the previous work [4], ELL was introduced as a format of sparse matrix storage. 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) operations.

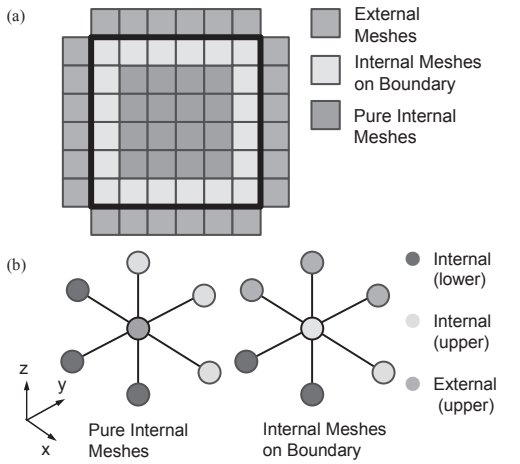

Fig. 8. Local data structure of pGW3D-FVM. (a) Internal/external meshes; (b) Upper/lower components of internal meshes

In Fig. 8, (a) and (b) provide the local data structure of pGW3D-FVM. The numbering of *external* meshes on each distributed mesh starts after all *internal* meshes in Fig. 8(a) are numbered. In the structured mesh in pGW3D-FVM, the initial numbering is *lexicographical*. Therefore, each *pure internal mesh* has three lower and three upper triangular components. But, an *internal mesh on a domain boundary* may have up to six upper triangular components, as shown in Fig. 8(b). In [4], the ELL format is applied to pGW3D-FVM with Cuthill-McKee (CMK) reordering, because CMK does not change the inequality relationship of the ID of each mesh for the structured meshes in pGW3D-FVM. In the ELL format, 2D-type arrays for matrices (e.g. `AMAT(6,N)`) are applied.

```
      do icol= NHYP(lev), 1, -1
        if (mod(icol,2).eq.1) then
!$omp parallel do private (ip,icel,j,SW)   for Boundary Cells
          do ip= 1, PEsmpTOT
          do icel= SMPindex(icol-1,ip,lev)+1, SMPindex(icol,ip,lev)
            SW= 0.0d0
            do j= 1, 6
              SW= SW + AUnew(j,icel)*Rmg(IAUnew(j,icel))
            enddo
            Rmg(icel)= Rmg(icel) - SW*DDmg(icel)
          enddo
          enddo
        else
!$omp parallel do private (ip,icel,j,SW)   for Pure Internal Cells
          do ip= 1, PEsmpTOT
          do icel= SMPindex(icol-1,ip,lev)+1, SMPindex(icol,ip,lev)
            SW= 0.0d0
            do j= 1, 3
              SW= SW + AUnew(j,icel)*Rmg(IAUnew(j,icel))
            enddo
            Rmg(icel)= Rmg(icel) - SW*DDmg(icel)
          enddo
          enddo
        endif                          IAUnew (6,N), AUnew (6,N)
      enddo
```

Fig.9. Procedures for backward substitution by original ELL format [4], Cache is not well-utilized for loops of pure internal meshes

```
      do icol= NHYP(lev), 1, -1
        if (mod(icol,2).eq.1) then
!$omp parallel do private (ip,icel,j,SW)   for Boundary Cells
          do ip= 1, PEsmpTOT
          do icel= SMPindex(icol-1,ip,lev)+1, SMPindex(icol,ip,lev)
            SW= 0.0d0
            do j= 1, 6
              SW= SW + AUnew6(j,icel)*Rmg(IAUnew6(j,icel))
            enddo
            Rmg(icel)= Rmg(icel) - SW*DDmg(icel)
          enddo
          enddo
        else
!$omp parallel do private (ip,icel,j,SW)   for Pure Internal Cells
          do ip= 1, PEsmpTOT
          do icel= SMPindex(icol-1,ip,lev)+1, SMPindex(icol,ip,lev)
            SW= 0.0d0
            do j= 1, 3
              SW= SW + AUnew3(j,icel)*Rmg(IAUnew3(j,icel))
            enddo
            Rmg(icel)= Rmg(icel) - SW*DDmg(icel)
          enddo
          enddo                        IAUnew3(3,N), AUnew3(3,N)
        endif                          IAUnew6(6,N), AUnew6(6,N)
      enddo
```

Fig.10. Procedures for backward substitution by *sliced* ELL format, `IAUnew/AUnew` are stored in separate arrays for boundary meshes and pure internal meshes, Cache is well-utilized for loops of pure internal meshes

Figure 9 shows the procedures of backward substitution of IC(0) smoothing for ELL-CGA [3]. Arrays for column ID's (`IAUnew(6,N)`), and coefficients (`AUnew(6,N)`) for upper triangular components of IC(0) matrix are used for both of pure internal meshes and internal meshes on boundary (*boundary meshes*) in Fig.8(b). At each level of Cuthill-McKee (CMK) reordering (`icol` in Fig.9), pure internal meshes and boundary meshes are calculated separately. Because reordering on a same level does not affect convergence [2,3,4], computations related to upper triangular components such as the backward substitution process in IC(0) smoothing for boundary meshes (with up to 6 upper components) are done first, then pure internal meshes (with up to 3) are calculated. This implementation of ELL provides 25% improvement of performance compared to CRS for HB 8×2 at 4,096 nodes, as shown in Fig.5. But, this approach is not necessarily efficient. `IAUnew/AUnew(4,N)-(6,N)` are not used for calculation of pure internal meshes, although these are stored on cache. In the present work, improved version of ELL (*Sliced ELL*) is introduced, where `IAUnew/AUnew` for pure internal meshes and boundary meshes are stored in separate arrays, `IAUnew3/AUnew3(3,N)` and `IAUnew6/AUnew6(6,N)`, as shown in Fig.10. This idea is very similar to *Sliced ELLPACK* proposed in [12]. In the *Sliced ELLPACK*, a sparse matrix is first divided into sub-matrices (*slices*) consisting of *S* non-zero off-diagonal components (*S*=1,…, N), and each slice is then stored in the ELL format. Thus, the redundancy

inherent in the ELL format is eliminated, and cache is well-utilized in loops for pure internal meshes. In the present work, we have two slices for *boundary meshes* and *pure internal meshes*, as shown in Fig.11. CMK reordering is also applied for the new format of matrix storage based on *sliced* ELL. The effect of formats of sparse matrix storage was evaluated using a single node (16 cores) of Fujitsu FX10 for *flat MPI*. The number of meshes per core was 262,144 (=64^3), and the total problem size was 4,193,304. Table 1 shows the results of the performance analyzer for Fujitsu FX10 [5]. Performance of multigrid process was evaluated. Performance of *sliced* ELL is 23% larger than that of original ELL due to better utilization of cache and effect of prefetching. Current version of the code is not efficient, because the code stores two series of arrays `IAUnew3/AUnew3(3,N)` and `IAUnew6/AUnew6(6,N)`, where extra memory space is required.

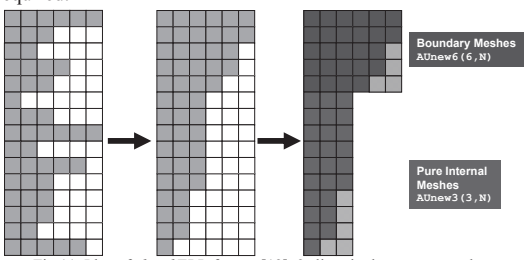

Fig.11. Idea of *sliced* ELL format [12], 2 slices in the present work

TABLE 1: PERFORMANCE OF MULTIGRID PROCEDURE, MGCG SOLVER FOR 262,144 MESHES (=64^3) WITH A SINGLE NODE (16 CORES) OF FUJITSU FX10 BY A PERFORMANCE ANALYZER [5], TOTAL PROBLEM SIZE IS 4,194,304

	Instruction	L2 Cache Miss	SIMD Operation Ratio (%)	GFLOPS
CRS	1.53×10^9	1.67×10^7	30.14	6.05
Original ELL (Fig.9)	4.91×10^8	1.27×10^7	93.88	6.99
Sliced ELL (Fig.10)	4.91×10^8	9.14×10^6	93.88	8.56

G. hCGA for Parallel Communication

CGA (Coarse Grid Aggregation) [3] provides efficiency and robustness of parallel multigrid method, as shown in Fig.5. Problem of this approach is coarse grid problem is solved on a single MPI process. Therefore, computational amount for coarse grid solver increases as number of total cores increases, as shown in Fig.7. This is more significant, if number of threads for each MPI process is smaller, such as flat MPI, HB 4×4. In the present work, hierarchical version of CGA (*h*CGA) is introduced, as shown in Fig.12. In *h*CGA, number of MPI processes is reduced and processes are repartitioned in a intermediate level before the final coarse grid solver on a single MPI process. For example, number of MPI processes is reduced from 12 to 3 at *level=m-3*, and switched to the final coarse grid solver at *level=m-2*, as shown in Fig.12. *h*CGA is also expected to reduce communication overhead at coarser levels of multigrid process. This type of approach is already introduced in recent works of parallel multigrid method on peta-scale supercomputers with more than 10^5 cores, such as IBM BlueGene/P, and Cray-XE6 systems [7,8]. Both of [7] and [8] adopted only *flat MPI*, and they dynamically reduce the number of MPI processes in order to ensure that overhead of parallel communication does not dominate the computation. Both of them define the minimum number of elements for each process below which the communication costs start to dominate. This critical value is defined as 2,000 in [7] and 1,000 in [8]. In the present work, we define the switching level according to the optimum one for CGA. If the optimum switching level for CGA is lev_{CGAopt}, (=7 in the case of Fig.5), the switching level to reduced number of processes (lev_{hCGA}) is set to be $lev_{hCGA} = lev_{CGAopt}$ -1, or =lev_{CGAopt} -2. If we consider the case in Fig.5, lev_{hCGA} is set to 5 or 6. Moreover, we have to specify the switching level to the final coarse grid solver (=$lev_{hCGA\text{-}CGA}$). In the present work, optimum value of lev_{hCGA} and $lev_{hCGA\text{-}CGA}$ are defined through empirical studies. Finally, number of MPI processes after repartitioning (PE_{rep}) is defined as 1/64 or 1/512 of the original number of MPI processes. PE_{rep} is also decided empirically.

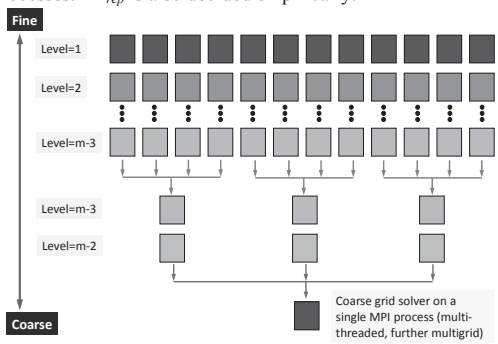

Fig.12. Procedures of hierarchical CGA (*h*CGA), where number of MPI processes is reduced before the final coarse grid solver of CGA on a single MPI process, In this case, number of MPI processes is reduced from 12 to 3 at *level=m-3*, and switched to the final coarse grid solver at *level=m-2*

IV. RESULTS

A. Overview

Optimization procedures for serial and parallel communications in parallel multigrid described in the previous chapter have been implemented for MGCG solver of pGW3D-FVM [2,3,4]. Table 2 summarizes details of each case. Four types of parallel programming models (flat MPI, HB 4×4, HB 8×2, and HB 16×1) are evaluated. Results of **C3** and **C4** are new in the present work.

TABLE 2: SUMMARY OF DETAILS OF EACH CASE

	Format of Matrix Storage	CGA/*h*CGA
C0 [3]	CRS	NO CGA (Fig.2)
C1 [4]	ELL (Fig.9)	
C2 [4]		CGA (Fig.3)
C3	*Sliced* ELL (Fig.10)	
C4		*h*CGA (Fig.12)

B. Weak Scaling

The performance of weak scaling was evaluated by using 8 to 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.

Figure 13 compares four types of parallel programming models for **C3**. Switching level to the coarse grid solver in CGA is set to 7 for all of HB 4×4, HB 8×2 and HB 16×1, which provides the best performance at 4,096 nodes for all cases. Each of HB 4×4, HB 8×2 and HB 16×1 provides excellent scalability up to 4,096 nodes, although HB 8×2 is slightly faster than others. Flat MPI is much slower than other hybrid parallel programming models with more than 10^4 cores because of overhead of coarse grid solver. Figure 14 provides the performance of **C0**-**C3** for HB 8×2. Switching level to the coarse grid solver in CGA (**C2** and **C3**) is set to 7. Improvement of performance by *sliced* ELL over ELL (improvement from **C2** to **C3**) at 4,096 nodes is 25%-31%, and 28% for HB 8×2. Effect of *sliced* ELL is significant.

Figure 15 compares best results of **C3** and **C4** at 4,096 nodes. Effect of *h*CGA is not so significant except for flat MPI where **C4** is 1.61 times faster than **C3**. **C4** is slightly faster for HB 4×4, while **C4** is slower than **C3** for HB 8×2 and HB 16×1 due to the overhead of repartitioning for *h*CGA. 2-digit number in each label after a colon on the transverse axis of Fig.15 indicates the number of iterations until convergence. This means that *h*CGA can make changes on convergence. Convergence of flat MPI and HB 4×4 is improved by *h*CGA.

Table 3 summarizes comparison between **C3** (CGA) and **C4** (*h*CGA) for *flat MPI* and HB 4×4. Optimum number of MPI processes after repartitioning (PE_{rep}) for each case is 128 processes (8 nodes) for *flat MPI*, and 256 processes (64 nodes) for HB 4×4, respectively. Optimum level for switching to reduced number of MPI processes for *h*CGA ($lev_{hCGAopt}$) is 6 for both of *flat MPI* and HB 4×4, while lev_{CGAopt} for flat MPI is 7, and that of HB 4×4 is 8. Critical mesh number per MPI process for repartitioning is 1,000-2,000 in the existing works [7,8], but this value for the optimum case of *flat MPI* is 8 in the present work, where number of meshes in each MPI process is equal to 8 (=2^3) at *lev*=6 as shown in Table 3. Overhead for coarse grid solver was significant for flat MPI with more than 10^4 cores, as shown in Fig.7. At 4,096 nodes, intermediate level of this coarse grid problem was solved by 8 nodes (128 MPI processes (=PE_{rep})) for **C4**, instead of by a single core for **C2** and **C3**.

Figure 16 compares best results of **C3** and **C4** for four parallel programming models. All of them except flat MPI (**C3**) are scalable up to 4,096 nodes, although HB 8×2 (**C3**) is slightly faster than others. Finally, Fig.17 provides the performance of **C3** and **C4** for the four parallel programming models at 64, 512, and 4,096 nodes. Generally speaking, **C4** with *h*CGA shows significant improvement of performance of *flat MPI*, as number of nodes increases.

TABLE 3: COMPARISON OF C3 (CGA) AND C4(*h*CGA) AT 4,096 NODES, FOR FLAT MPI AND HB 4×4

		lev_{CGAopt} $lev_{hCGAopt}$	$lev_{hCGA\text{-}CGAopt}$ (PE_{rep})	Iter's	sec.
Flat MPI	C3	7	-	64	13.2
	C4	6	3 (128 proc's 8 nodes)	61	8.22
HB 4×4	C3	8	-	59	8.08
	C4	6	2 (256 proc's, 64 nodes)	56	7.97

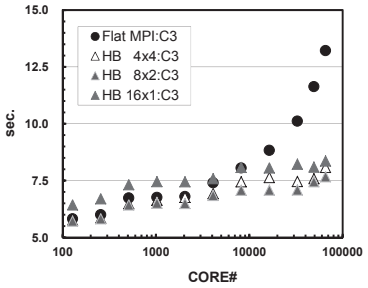

Fig. 13. Performance of MGCG solver on Fujitsu FX10 using up to 4,096 nodes (65,536 cores), weak scaling: 262,144 (=64^3) meshes/core, max. total problem size: 17,179,869,184 meshes, elapsed computation time for MGCG, best results of *sliced* ELL with CGA (**C3**), switching to coarse grid solver at *level*=7 for CGA

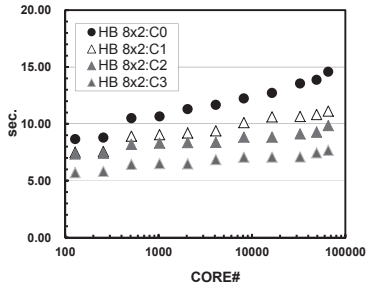

Fig. 14. Performance of MGCG solver on Fujitsu FX10 using up to 4,096 nodes (65,536 cores), weak scaling: 262,144 (=64^3) meshes/core, max. total problem size: 17,179,869,184 meshes, elapsed computation time for MGCG, HB 8×2, **C3** (*sliced* ELL) is 28% faster than **C2** (ELL) at 4,096 nodes

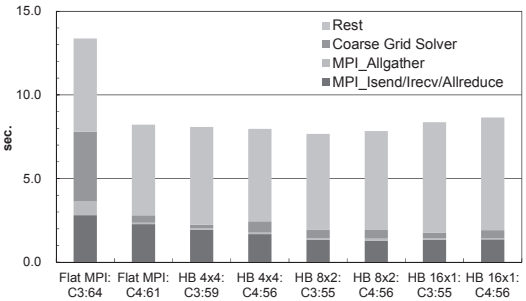

Fig. 15. Performance of MGCG solver on Fujitsu FX10 using 4,096 nodes (65,536 cores), total problem size: 17,179,869,184 meshes, comparison of **C3** and **C4**, effect of *h*CGA

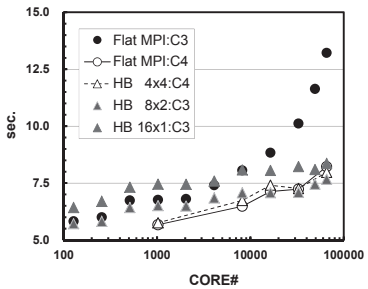

Fig. 16. Performance of MGCG solver on Fujitsu FX10 using up to 4,096 nodes (65,536 cores), weak scaling: 262,144 (=64^3) meshes/core, max. total problem size: 17,179,869,184 meshes, elapsed computation time for MGCG, best results of **C3** and **C4**, and flat MPI (**C3**)

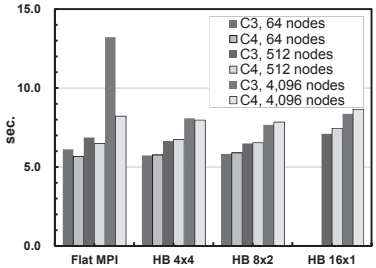

Fig. 17. Performance of MGCG solver on Fujitsu FX10 using up to 4,096 nodes (65,536 cores), weak scaling: 262,144 (=64^3) meshes/core, max. total problem size: 17,179,869,184 meshes, elapsed computation time for MGCG

C. 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 to 4,096 nodes of Fujitsu FX10. At 4,096 nodes, the problem size per each core is only 4,096 meshes (=16^3). Figure 18 shows the performance of the MGCG solver, and compares **C3** (*sliced* ELL with CGA) and the best case of **C4** (*sliced* ELL with *h*CGA) for each parallel programming model. The performance of *flat MPI* (**C3**) with 8 nodes (128 cores) is 100%, and the parallel performance is 50%–67% up to 512 nodes (8,192 cores). At 4,096 nodes, *flat MPI* (**C4**) provides the best performance (14.6%). Improvement of performance of flat MPI by *h*CGA (from **C3** to **C4**) is significant, and the ratio of performance is 6.27 times at 4096 nodes. Improvement of performance by *sliced* ELL over original ELL (from **C2** to **C3**) at 4,096 nodes is 21.8% (flat MPI), 14.2% (HB 4×4), 15.3% (HB 8×2), and 9.1% (HB 16×1), respectively.

D. Improvement of Parallel Communication by hCGA

In this section, effect of *h*CGA on improvement of parallel communication is evaluated for both of weak and strong cases. Figure 19 compares overhead for parallel communications per a single CG iteration for weak scaling computation at 4,096 nodes. Overhead for parallel communication includes, MPI_Allreduce for dot products, MPI_Isend/Irecv/Waitall for communications at boundaries, and memory copies at boundaries. Generally speaking, *h*CGA improve the parallel communications of MGCG solver. Effect of improvement is 15% for flat MPI, and less than 1% for HB 16×1. Finally, Fig.20 compares parallel communication overhead for strong scaling computing at 4,096 nodes. Effect of improvement by *h*CGA is more than that for weak scaling cases, where effect of improvement is 35% for flat MPI, and 8% for HB 16×1.

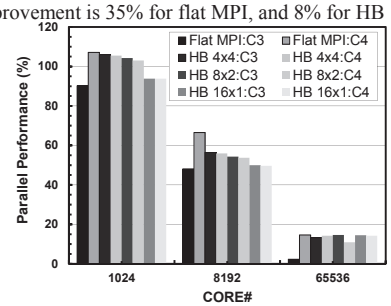

Fig. 18. Performance of MGCG solver on Fujitsu FX10 using up to 4,096 nodes (65,536 cores), strong scaling: 268,435,456 meshes (=1024×512×512), parallel performance based on the performance of *flat MPI* (**C3**) with 8 nodes (128 cores)

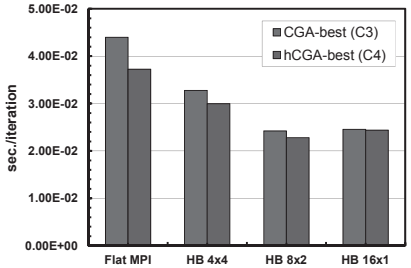

Fig. 19. Effect of *h*CGA on overhead of parallel communications in MGCG solver, overhead for parallel communications per a single CG iteration, total problem size: 17,179,869,184 meshes at 4,096 nodes

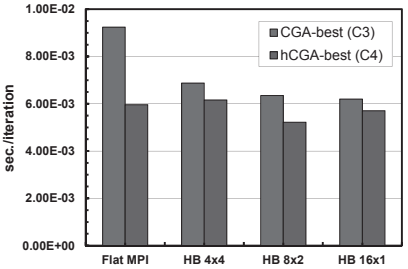

Fig. 20. Effect of *h*CGA on overhead of parallel communications in MGCG solver, overhead for parallel communications per a single CG iteration, total problem size: 268,435,456 meshes at 4,096 nodes

V. Summary and Future Work

The parallel multigrid and MGCG include both of serial and parallel communication processes which are generally expensive. In the present work, new format for sparse matrix storage based on *sliced* ELL is proposed for optimization of

serial communication on memories, and hierarchical coarse grid aggregation (*h*CGA) is introduced for optimization of parallel communication by message passing. The proposed methods are implemented for pGW3D-FVM, and the robustness and performance of the code was evaluated using up to 4,096 nodes (65,536 cores) of the Fujistu FX10 system. The parallel MGCG solver using the *sliced* ELL format provided performance improvement in both weak scaling (25%–31%) and strong scaling (9%–22%) compared to the code using the original ELL format [4]. Moreover, *h*CGA provided excellent performance improvement in both weak scaling (1.61 times) and strong scaling (6.27 times) for *flat MPI* parallel programming model. *h*CGA was also effective for improvement of parallel communications. Effect of *sliced* ELL on serial communication was significant, while that of *h*CGA on parallel communication was not so impressive except for *flat MPI* cases. Because *h*CGA proved to be very effective for reducing overhead of coarse grid solver, that will also provide more significant effect on hybrid parallel programming models with larger number of nodes. As was mentioned in Chapter II, computational amount of coarse grid solver for each core of *flat MPI* is 256 (=16×16) times as large as that of HB 16×1. Therefore, *h*CGA is expected to be really effective for HB 16×1 with more than 2.50×10^5 nodes (4.00×10^6 cores) of Fujitsu FX10, where the peak performance is more than 60 PFLOPS.

In the present work, we focused on optimization of communications with neighboring MPI processes at process boundaries by MPI_Isend/Irecv/Waitall as *parallel communication*, but overhead by collective communication (e.g. MPI_Allreduce) is more significant with more than 2.50×10^5 nodes. Figure 21 shows overhead by a single MPI_Allreduce call for sending a single scalar variable for dot product in the weak scaling case of MGCG solver, with up to 4,096 nodes of Fujitsu FX10. In order to reduce overhead by this type collective communication, communication avoiding/hiding type of algorithm combined with MPI functions for non-blocking collective communications supported in MPI 3.0 specification [13] is also effective.

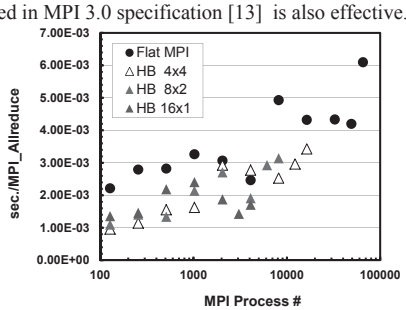

Fig. 21. Overhead by MPI_Allreduce for dot product (a single scalar variable) of MGCG solver on Fujitsu FX10 using up to 4,096 nodes (65,536 cores), weak scaling: 262,144 (=64^3) meshes/core, max. total problem size: 17,179,869,184 meshes, best results of **C3** and **C4**

More efficient storage of *sliced ELL* arrays and optimization of *h*CGA function are the critical issues for future work. Especially, improvement of communication procedures for repartitioning of MPI processes is important. CGA and *h*CGA include a various types of parameters, and the optimum values of those were derived through empirical studies in the present work. Development of methods for automatic selection of these parameters [14] is also an interesting technical issue for future work. Optimum parameters can be estimated based on calculation of computational amounts, performance models, parameters of hardware, and some measured performance of the system. But it is not so straightforward. Because some of these parameters also make effects on convergence, construction of such methods for automatic selection is really challenging.

ACKNOWLEDGMENT

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] HPCG: High Performance Conjugate Gradients: https://software.sandia.gov/hpcg/
[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 & Applications for Multi/Manycores (2012)
[3] 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) 199-206
[4] Nakajima, K., Large-scale Simulations of 3D Groundwater Flow using Parallel Geometric Multigrid Method, Procedia Computer Science 18 (2013) 1265-1274
[5] Information Technology Center, The University of Tokyo: http://www.cc.u-tokyo.ac.jp/
[6] Baker, A., T. Gamblin, M. Schultz, U. Yang, Challenge of Scaling Algebraic Multigrid across Modern Multicore Architectures, Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium (IPDPS'11) (2011) 275-286
[7] Lin, P.T., Improving multigrid performance for unstructured mesh drift-diffusion simulations on 147,000 cores, International Journal for Numerical Methods in Engineering 91 (2012) 971-989
[8] Sundar, H., G. Biros, C. Burstedde, J. Rudi, O. Ghattas, and G. Stadler, Parallel Geometric-Algebraic Multigrid on Unstructured Forests of Octrees, ACM/IEEE Proceedings of the 2012 International Conference for High Performance Computing, Networking, Storage and Analysis (SC12) (2012)
[9] Fujitsu: http://www.fujitsu.com/
[10] Deutsch, C.V. and A.G. Journel, GSLIB Geostatistical Software Library and User's Guide, Second Edition. Oxford University Press (1998)
[11] Smith, B., P. Bjørstad, and W. Gropp, *Domain Decomposition, Parallel Multilevel Methods for Elliptic Partial Differential Equations*, Cambridge Press (1996)
[12] Monakov, A., A. Lokhmotov, and A. Avetisyan, Automatically tuning sparse matrix-vector multiplication for GPU architectures, Lecture Notes in Computer Science 5952 (2010) 112-125
[13] Ghysels, P. and W. Vanroose, Hiding global synchronization latency in the preconditioned Conjugate Gradient algorithm, Parallel Computing 40-7 (2014) 224-238
[14] Nakajima, K., Automatic Tuning of Parallel Multigrid Solvers using OpenMP/MPI Hybrid Parallel Programming Models, Lecture Notes in Computer Science 7851 (2013) 435-450