Parallel Multilevel Iterative Linear Solvers with Unstructured Adaptive Grids for Simulations in Earth Science

Kengo Nakajima

Department of Computational Earth Sciences, Research Organization for Information Science and Technology (RIST), Tokyo, Japan (e-mail: nakajima@tokyo.rist.or.jp, phone: +81-3-3712-5321, fax: +81-3-3712-5552).

Abstract

A new multigrid-preconditioned conjugate gradient (MGCG) iterative method for parallel computers is presented. Iterative solvers with preconditioning, such as the incomplete Cholesky or incomplete LU factorization methods, represent some of the most powerful tools for large-scale scientific computation. However, the number of iterations required for convergence by these methods increases with the size of the problem. In multigrid solvers, the rate of convergence is independent of problem size, and the number of iterations remains fairly constant. Multigrid is also a good preconditioning algorithm for Krylov iterative solvers. In this study, the MGCG method is applied to Poisson equations in the region between 2 spherical surfaces on semi-unstructured, adaptively generated prismatic grids, and to grids with local refinement. Computations using this method on a Hitachi SR2201 with up to 128 processors demonstrated good scalability.

1. Introduction

In many large-scale scientific simulation codes, the majority of computation is devoted to linear solvers. Preconditioned Krylov iterative solver such as conjugate gradient method with incomplete Cholesky factorization preconditioning (ICCG) [1] provides robust convergence for a wide range of scientific applications. Incomplete Cholesky (IC) and incomplete LU (ILU) factorizations involve globally dependent operations, yet can be localized for parallel computation[2] and provide smooth convergence. However, ICCG-based solvers are not *scalable* because the number of iterations required for convergence increases with the size of the problem. This becomes critical when solving problems with $>10^9$ degrees of freedom (DOF) on $>10^4$ processors.

Multigrid[3] is a well-known scalable method for which the rate of convergence is independent of problem size, and the number of iterations remains fairly constant. Multigrid is also a good preconditioning algorithm for Krylov iterative solvers. Multigrid solvers and preconditioners have been widely used in finite-difference methods with structured grids since the mid 1980s, however multigrid is not popular for finite-element methods involving unstructured grids. Recently, various multigrid methods for unstructured grids have been developed[4][5][6][7][8], for both parallel and serial computers.

In this study, a multigrid-preconditioned conjugate gradient iterative method (MGCG) on parallel computers was developed and applied to Poisson equations in the region between 2 spherical surfaces on adaptively generated semi-unstructured prismatic grids based on the method in [8]. This procedure has also been applied to grids with local refinement.

2. Multigrid Method : Overview

Multigrid is a scalable method for solving linear equations. Relaxation methods such as Gauss-Seidel efficiently damp high-frequency error but do not eliminate low-frequency error. The multigrid approach was developed in recognition that this low-frequency error can be accurately and efficiently solved on a coarser grid. This concept is explained here in the following simple 2-level preconditioning method, as described in [4]. If we have obtained the following linear system on a fine grid :

$$A_F\, u_F = f$$

and A_C as the discrete form of the operator on the coarse grid, a simple coarse grid correction can be given by :

$$u_F^{(i+1)} = u_F^{(i)} + R^T A_C^{-1} R\,(\,f - A_F\, u_F^{(i)}\,)$$

where R^T is the matrix representation of linear interpolation from the coarse grid to the fine grid (*prolongation* operator) and R is called the restriction operator. Thus, it is possible to calculate the residual on the fine grid, solve the coarse grid problem, and interpolate the coarse grid solution on the fine grid. This process can be described as follows :

1. Relax the equations on the fine grid and obtain the result $u_F^{(i)} = S_F\,(\,A_F, f\,)$. This operator S_F (e.g., Gauss-Seidel) is called the *smoothing operator*.
2. Calculate the residual term on the fine grid by $r_F = f - A_F\, u_F^{(i)}$.
3. *Restric*t the residual term on to the coarse grid by $r_C = R\, r_F$.
4. Solve the equation $A_C\, u_C = r_C$ on the coarse grid ; the accuracy of the solution on the coarse grid affects the convergence of the entire multigrid system[3][4][5].
5. Interpolate (or *prolong*) the coarse grid correction on the fine grid by $\Delta u_C^{(i)} = R^T u_C$.
6. Update the solution on the fine grid by $u_F^{(i+1)} = u_F^{(i)} + \Delta u_C^{(i)}$.

Recursive application of this algorithm for 2-level preconditioning for consecutive systems of coarse-grid equations gives a multigrid V-cycle[3] (Fig. 1). If the components of the V-cycle are defined appropriately, the result is a method that uniformly damps all frequencies of error with a computational cost that depends only linearly on the problem size. In other words, multigrid algorithms are *scalable*.

In the V-cycle, starting with the finest grid, all subsequent coarser grids are visited only once. In the down-cycle, smoothers damp oscillatory error components at different grid scales. In the up-cycle, the smooth error components remaining on each grid level are corrected using the error approximations on the coarser grids.[4] Alternatively, in a W-cycle[3] (Fig. 1), the coarser grids are solved more rigorously in order to reduce residuals as much as possible before going back to the more expensive finer grids.

Multigrid algorithms tend to be problem-specific solutions and less robust than preconditioned Krylov iterative methods such as the IC/ILU methods. Fortunately, it is easy to combine the best features of multigrid and Krylov iterative methods into one algorithm ; multigrid-preconditioned Krylov iterative methods. The resulting algorithm is robust, efficient and scalable.

One of the most important issues in multigrid is the construction of the coarse grids. There are 2 basic multigrid approaches ; geometric[7][8] and algebraic[6]. In geometric multigrid, the geometry of the problem is used to define the various multigrid components. In contrast, algebraic multigrid methods use only the information available in the linear system of equations, such as matrix connectivity.

The convergence rate of standard multigrid methods degenerates on problems involving anisotropic discrete operators, such as those that appear in very thin meshes near the wall in Navier-Stokes computations. In these cases, the error becomes smooth in the *thin* direction where the connection is strong, but it is not smooth in the direction where the connection is weak. One popular approach adopted to deal with anisotropic operators is the use of *semi-coarsening*, where multigrid coarsening is applied adaptively in each coordinate direction. By coarsening the grid in a certain direction, anisotropy on the coarser grid can be reduced.

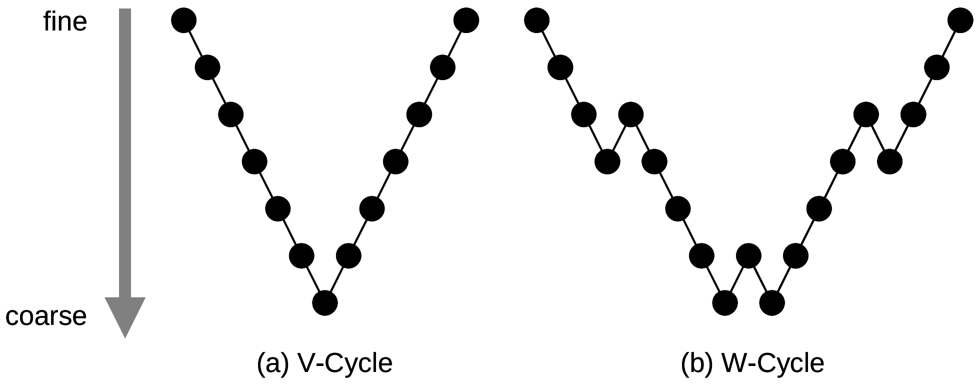


Fig. 1 V- and W-cycle for multigrid operation

3. Parallel Multigrid Preconditioned Iterative Solvers

3.1 Applications

In this study, the target application is 3D incompressible thermal convection in the region between 2 spherical surfaces. This geometry appears often in simulations in earth science for both fluid earth (atmosphere and ocean) and solid earth (mantle and outer core). Here, a semi-implicit pressure-correction scheme[9] is applied, in which momentum and energy equations are solved explicitly, and the pressure-correction Poisson equation :

$$\Delta\phi = \nabla \cdot u'$$

is solved for an incompressibility constraint. The Poisson equation solver is the most expensive process in this computation, and the convergence acceleration of this process is critical for increasing the overall speed of this method. In this study, the Poisson equation solver is the main consideration.

Semi-unstructured prismatic grids generated from triangles on a spherical surface are used (Fig. 2). Meshes are initiated from an icosahedron and are globally refined recursively as in Fig. 3. The grid hierarchy defined by recursive refinement can be utilized as coarse grid formation (Fig. 4). In the current application, velocity components and temperature are defined at cell corners, and pressure and potential for pressure correction are defined at cell centers. Therefore, the dependent variables are defined at the cell center in this study.

The surface of the model is covered with triangles, which provide geometric flexibility, while the structure of the mesh in the direction normal to the surface provides thin prismatic elements suitable for the viscous region.

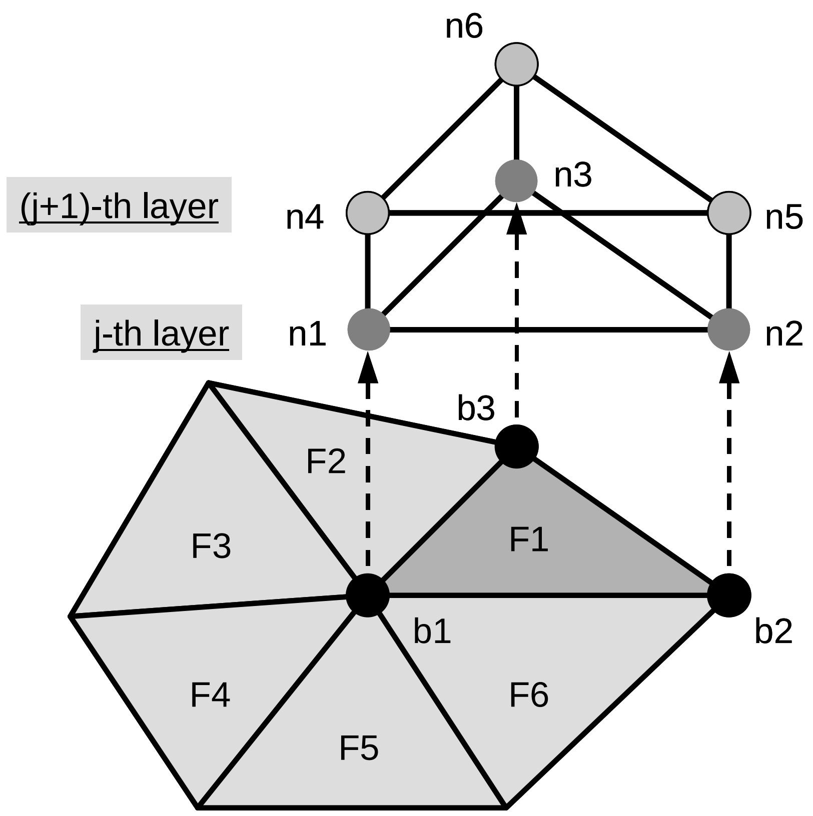
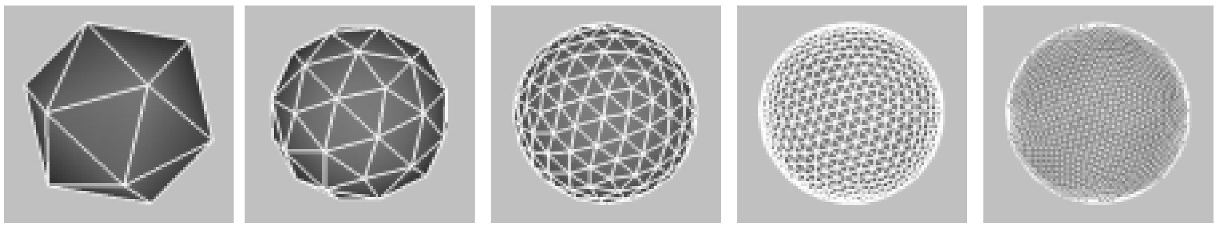


Fig. 2 Prisms generated from triangular facets



Level 0
12 nodes
20 tri's

Level 1
42 nodes
80 tri's

Level 2
162 nodes
320 tri's

Level 3
642 nodes
1,280 tri's

Level 4
2,562 nodes
5,120 tri's

Fig. 3 Surface triangle meshes generated from icosahedron
4 children generated from 1 parent triangle

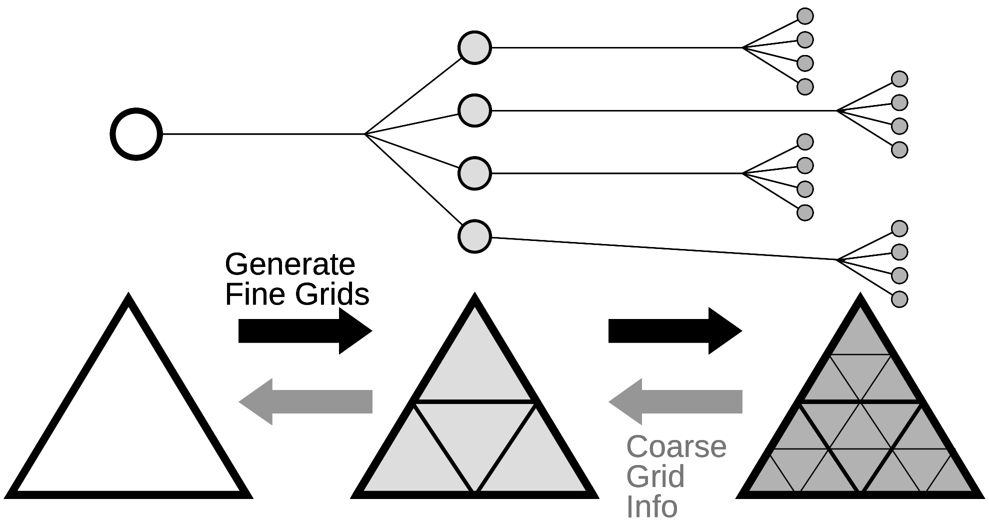


Fig. 4 Grid hierarchy from successive refinement

3.2 Parallel MGCG solvers for Poisson Equations

The parallel MGCG solver was implemented in Fortran 90 using MPI[10]. The features of the procedure are summarized as follows :

(1) V-cycle MGCG solver.
(2) Gauss-Seidel or ILU(0) smoothing.
(3) Semi-coarsening in lateral and normal-to-surface direction.
(4) Partition of entire region in radial (normal-to-surface) direction.
(5) Definition of multilevel communication tables at partition interfaces
(6) Includes consideration for the effect of local grid refinement

The Gauss-Seidel iterative method was adopted as the smoother, but ILU(0) factorization[1] was also tested as a smoother for the preconditioning component of the CG iterative method. The computational procedure for the preconditioned CG method for solving *Ax = b* is as follows[1] :

```
Compute $r^{(0)}=b - A x^{(0)}$ for some initial guess for $x^{(0)}$
for i= 1, 2, ...
   solve $M z^{(i-1)} = r^{(i-1)}$ where M is the preconditioner
   $\rho^{(i-1)}= r^{(i-1)T} z^{(i-1)}$
   if i= 1
      $p^{(1)}= z^{(0)}$
     else
      $\beta^{(i-1)}= \rho^{(i-1)}/ \rho^{(i-2)}$
      $p^{(i)} = z^{(i-1)} + \beta^{(i-1)} p^{(i-1)}$
   endif
   $q^{(i)}= A p^{(i)}$
   $\alpha^{(i)}= \rho^{(i-1)}/(p^{(i)} q^{(i)})$
   $x^{(i)}= x^{(i-1)} + \alpha^{(i)} p^{(i)}$
   $r^{(i)}= r^{(i-1)} - \alpha^{(i)} q^{(i)}$
   check convergence; continue if necessary.
end
```

The multigrid procedure is applied to solve the preconditioning matrix *Mz=r* where *M* is set identical to *A* in this study.

The V-cycle method described in Fig. 1 has been adopted. In each cycle, the Gauss-Seidel procedure is repeated 5 times for both restriction (*fine-to-coarse*) and prolongation (*coarse-to-fine*), or until convergence has stagnated, as is shown in Fig. 5. The ILU(0) smoother has been implemented with the additive Schwartz domain decomposition method[4] at each level. At each multigrid level, 2 iterations (i.e., 1 smoothing + 1 domain decomposition + 1 smoothing) are applied.

Semi-coarsening is applied in the lateral and normal-to-surface directions. In order to preserve the semi-unstructured grid features in the normal-to-surface direction, the entire region is partitioned in the radial direction.

The parallel multigrid cycle for coarsening is executed until :

- Lateral direction : Initial icosahedron (20 triangles)
- Normal-to-surface direction : 1 layer

on each processing element (PE). The multigrid procedure is then continued on a single PE until the number of layer is equal to 2. The equation on the coarsest grid ($20\times2=40$ cells) is solved accurately by the Gauss-Seidel method. These single-PE computations are very small and do not affect the parallel efficiency. As mentioned in the previous section, the accuracy of the solution at the coarsest level strongly affects the convergence of the entire multigrid system. If we choose *deeper* levels for the multigrid, the size of the problem at the coarsest level is reduced and a convergent solution can be obtained rapidly by Gauss-Seidel iterations. A very deep multigrid level is chosen for this reason.

In parallel computation with unstructured or semi-unstructured grids using the message passing library, the communication tables for partitions should be defined explicitly by the user[2]. In this study, multilevel communication tables were defined at each level according to the multigrid procedure (Fig. 6). As variables are defined at cell centers, the entire region is partitioned in a *cell-based* manner. Cells are classified into the following 3 categories from the viewpoint of message passing :

- Internal cells (originally assigned cells)
- External cells (cells that form the matrix connectivity in the partition but are located outside of the partition)
- Boundary cells (*external cells* of other partitions)

Values in *boundary* cells in the partitions are *sent* to the neighboring partitions and are *received* as *external* cells at the *destination* partition. This type of communication table is defined at each grid level.

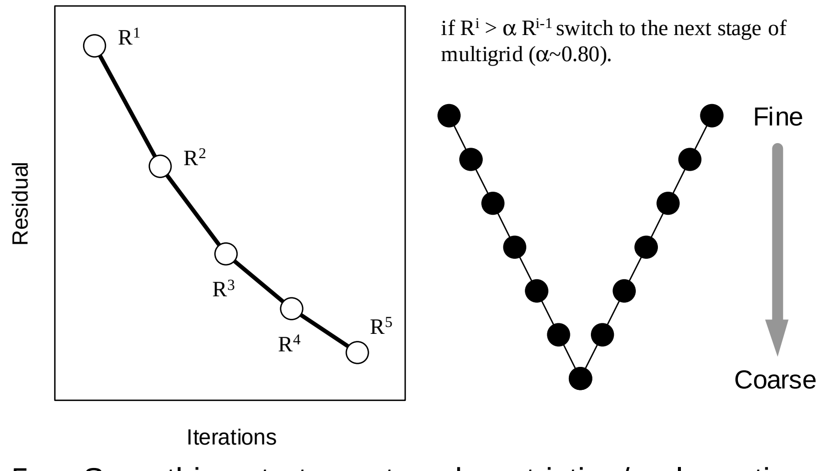

if $R^i > \alpha R^{i-1}$ switch to the next stage of multigrid (α~0.80).
R^1
R^2
R^3
R^4
R^5
Residual
Iterations
Fine
Coarse

Fig. 5 Smoothing strategy at each restriction/prolongation stage

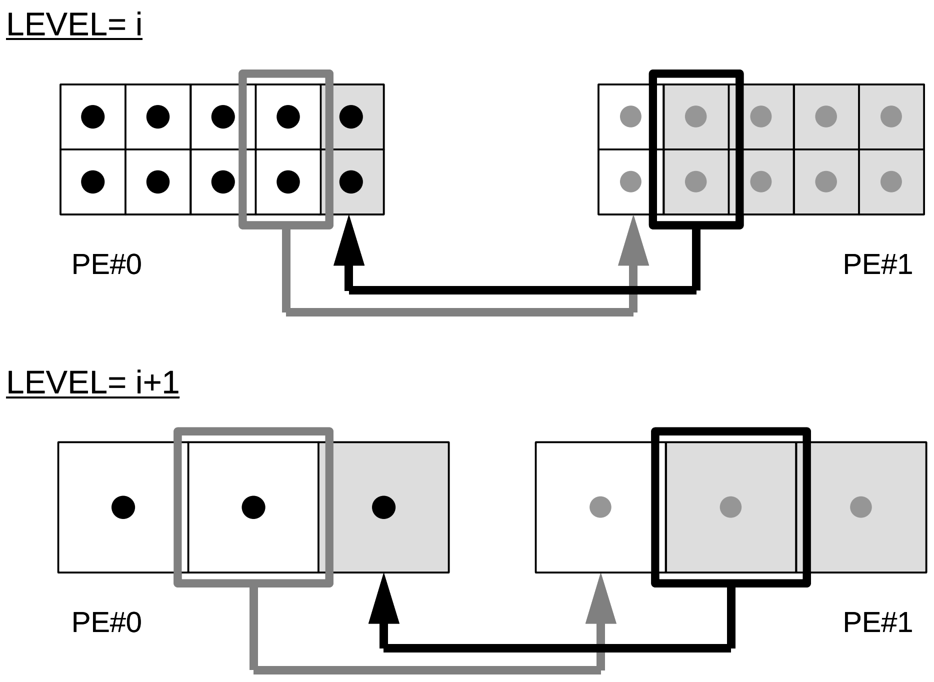


Fig. 6 Multilevel communication table

3.3 Grid Adaptation

Adaptive methods in applications involving unstructured meshes have evolved as efficient tools for obtaining numerical solutions without prior knowledge of the details of the underlying physics[8].

Here, a dynamic adaptation algorithm developed by the author for 3D unstructured meshes[8] is applied. The algorithm is capable of simultaneous refinement and coarsening of the appropriate regions in the flow domain.

The adaptation algorithm is guided by a feature detector that senses regions with significant changes in flow properties, such as shock waves, separations and wakes. Velocity differences and gradients are used for feature detection, and threshold parameters are set in order to identify the regions to be refined or coarsened. The details of the method used for feature detection in this study are described in [8]. In the present implementation, the feature detector marks edges.

The prisms are then refined directionally in order to preserve the structure of the mesh in the normal-to-surface direction. The prismatic mesh refinement proceeds by dividing only the *lateral* edges and faces, which are then refined by either *quadtree* or *binary* division. The resulting surface triangulation is replicated in each successive layer of the prismatic mesh as illustrated in Fig. 7. As is seen from this figure, the prismatic mesh refinement preserves the structure of the initial mesh in the direction normal to the surface.

In order to avoid excessive mesh skewness, repeated *binary* divisions of prisms are not allowed. Furthermore, in order to avoid sudden changes in mesh size, the mesh refinement algorithm also limits the maximum difference in embedding level between neighboring elements to less than 2.

Recently, various multigrid methods for locally refined grids have been developed for block-structured grids solved by finite-difference methods. The typical procedure described in [3] is to utilize the grid hierarchy for an adapted grid and to apply a nested multigrid procedure for each adaptation level. This approach (*level-by-level method*) usually requires additional memory and computations for *fine* cells without adaptation (white triangles in Fig. 8). In this study, we applied the *direct jump method*, where the solution starts from fine grid with *full* *(deepest)* adaptation level and then jumps back directly to the 2nd globally finest grid level in the multigrid procedure as is described in Fig. 8.

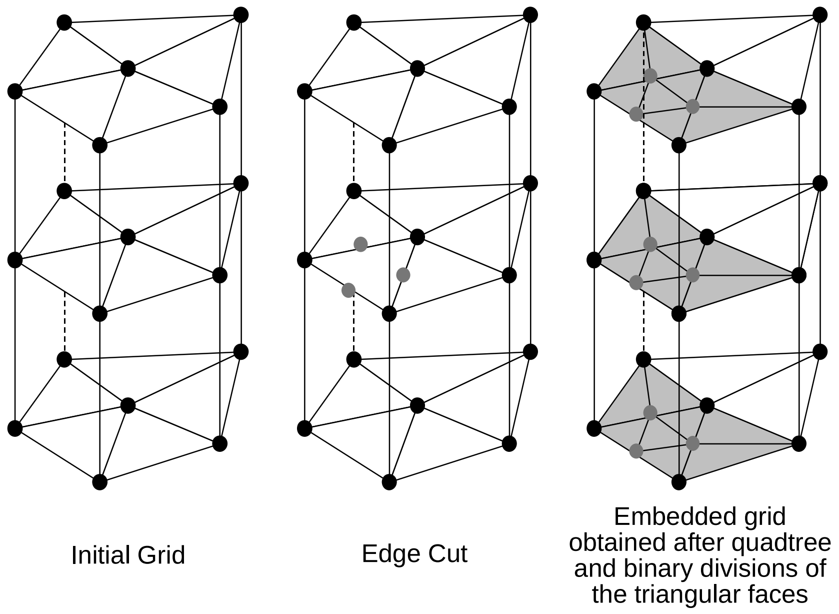

Initial Grid
Edge Cut
Embedded grid obtained after quadtree and binary divisions of the triangular faces

Fig. 7 Directional refinement of prisms based on *quadtree* and *binary* divisions of triangular faces

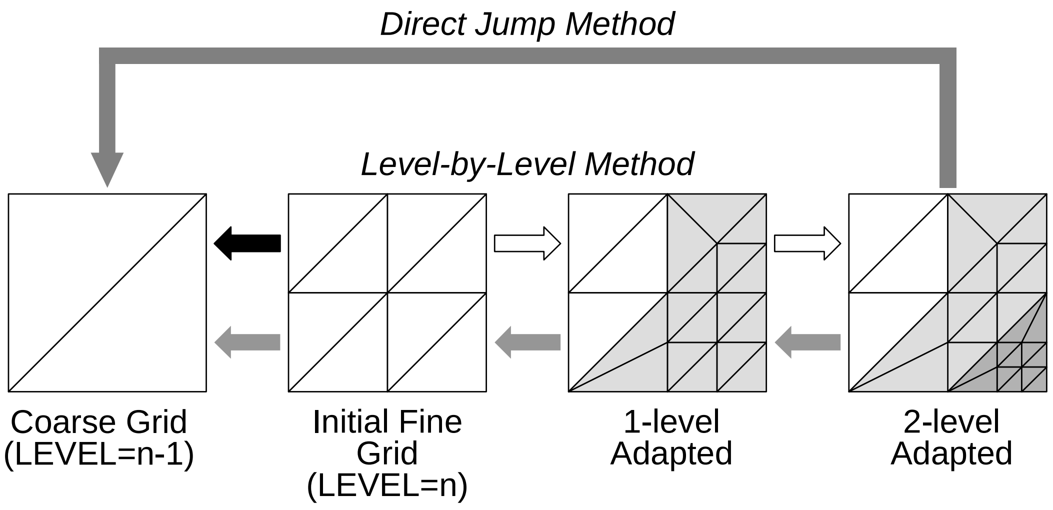


Fig. 8 Multigrid strategy for locally refined grids
Level-by-level and direct jump method

4. Results

The developed methods were tested on Poisson equations in the region between 2 spherical surfaces using a Hitachi SR2201 parallel computer at the University of Tokyo[11] with up to 128 processors. Here, 2 problems were considered. In both problems the following homogeneous Poisson equation was solved :

$$\Delta\phi = 1$$

In the 1st application, the problem size for 1 processor was fixed at 320 triangles (level=2 in Fig. 3) $\times$ 900 layers = 288,000 cells, and computations were performed using 2 to 128 PEs, corresponding to 576,000 to 36,864,000 cells.

The inner radius of the sphere is 0.50 units and the thickness of each layer was fixed at 0.01. Two different boundary conditions were defined, as follows :

- **Uniform** : Dirichlet boundary condition (ϕ=0) for all triangles on the outermost surface of the prisms.
- **Single-patch** : Dirichlet boundary condition (ϕ=0) for 1 triangle of the initial icosahedron on the outermost surface of the prisms. The Dirichlet boundary condition was applied to all children and grandchildren generated from this 1 triangle of the 20 ones that make up the initial icosahedron. This configuration produces very *ill-conditioned* coefficient matrices compared to the *uniform* cases.

Figures 9 (a) and (b) show the results (elapsed computation time including communication) for MGCG/GS (Gauss-Seidel) and ICCG computations in application-I. The computation time for MGCG/GS remains almost constant when a limited number of PEs are employed, but tends to increase with the number of PEs at higher degrees of parallelization. This tendency is attributed to the localization of Gauss-Seidel smoothing[2], approaching Jacobi smoothing as the number of PEs increases. However, even in the 128 PE cases, MGCG/GS is much faster than ICCG.

Figures 10 (a) and (b) show the results for MGCG/GS, MGCG/ILU(0) and ICCG for application-I with up to 32 PEs. MGCG/ILU(0) exhibits more robust convergence than MGCG/GS, particularly for the cases with the *single-patch* boundary condition. Figures 11 (a) and (b) show the iterations until convergence for these 3 methods. The number of iterations for MGCG/ILU(0) remains relatively constant compared to MGCG/GS, even when the number of PEs increases.

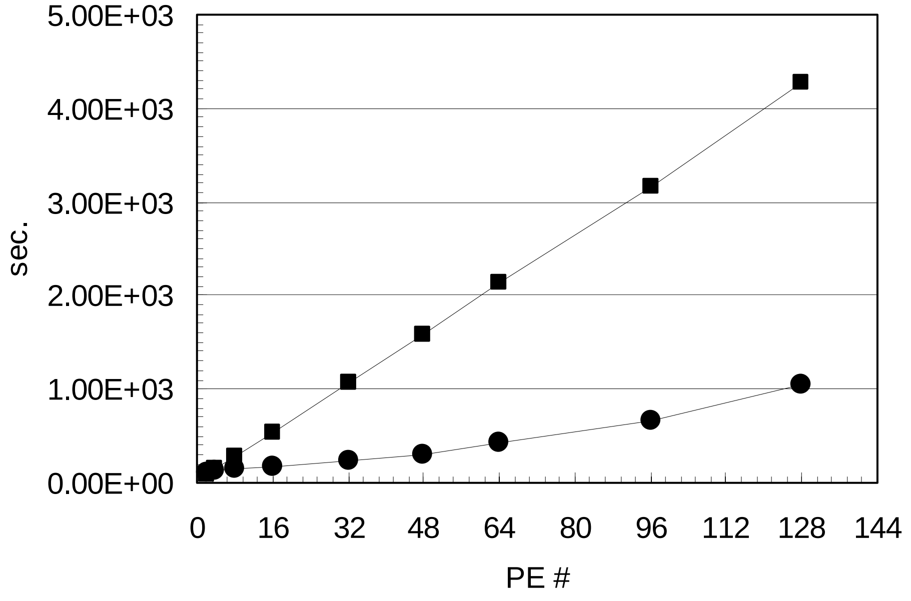
In the 2nd application, the effect of local grid refinement and the multgrid strategy (*direct jump* and *level-by-level*) were evaluated. The inner radius of the sphere is 0.50 units and the thickness of the each layer was fixed at 0.01. The 2 boundary conditions used in the 1st application were also applied to this problem. The initial grid was the level-2 grid (Fig. 3) with 320 triangles. As shown in Fig. 12, a 3-level grid adaptation was applied. At each adaptation level, the number of triangular facets varies as follows :

• 1st level:	532	triangles
• 2nd level:	1508	triangles
• 3rd level:	4448	triangles

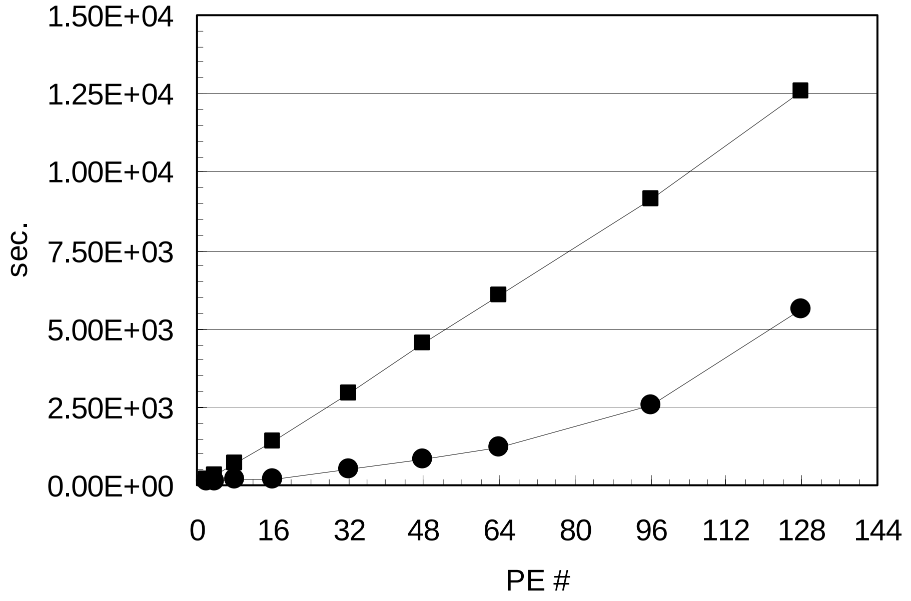
In the 2nd application, the number of layers on each PE was fixed at 50, and the Poisson equation examined in the 1st application was solved by MGCG/GS using between 2 and 32 PEs. Figure 12 shows the computation time normalized by the number of cells (i.e., degrees of freedom) on each processor according to problem size (i.e., PE number) using the *direct jump method*. If the procedure is scalable, the curves should be horizontal. Under the *uniform* boundary condition, MGCG/GS with the *direct jump method* provides very good scalability except for the 32 PE case with 3-level adapted grids. Under the *single-patch* boundary condition, the efficiency decreases as the number of PEs increases. This is similar to the results of the 1st application (Figs. 9-11).

Figure 13 shows a comparison between the *direct jump* and *level-by-level* multigrid strategies for an adapted grid under the *uniform* boundary conditions. If the adaptation level is *shallow*, the 2 methods are competitive, however at *deeper* levels of adaptation, the *direct jump method* provides much higher efficiency.

In both applications, the communication overhead is very low (less than 1%) even in the 128 PE cases, except for the cases with initial grids of 320 triangles in the 2nd applications (50 layers per PE) where the computational load is rather low.

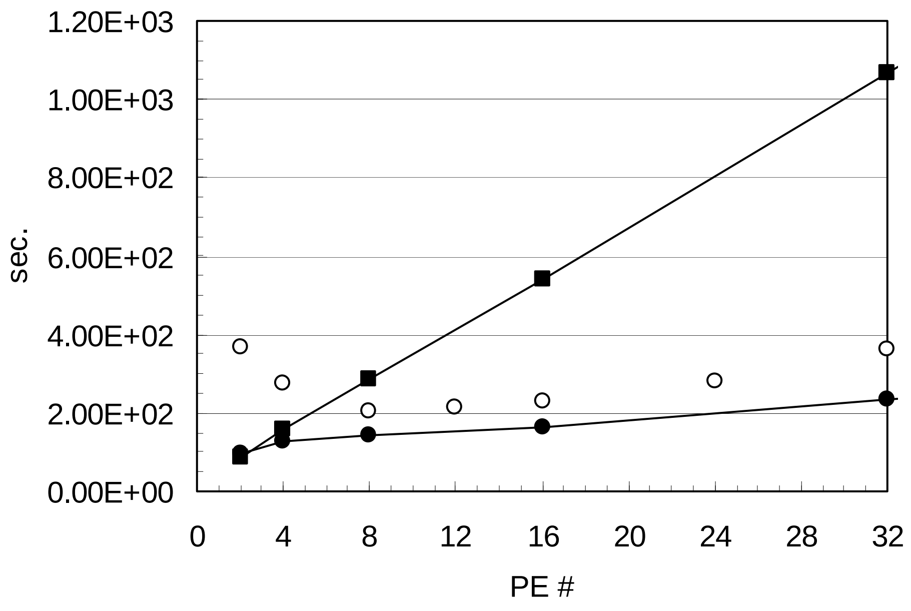


(a) *Uniform* boundary condition

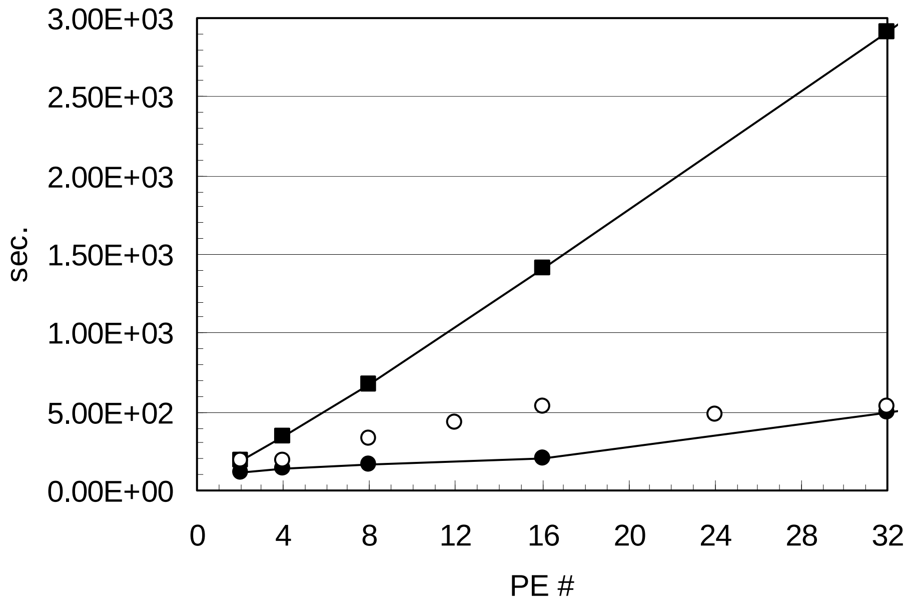


(b) *Single-patch* boundary condition

Fig. 9 Results of application-I. Computation time (including communication for parallel computing) for fixed problem size on each processor (320×900=288,000 cells/PE) for 2 to 128 PEs (up to 36,864,000 cells). Black squares: ICCG, black circles: MGCG/GS.

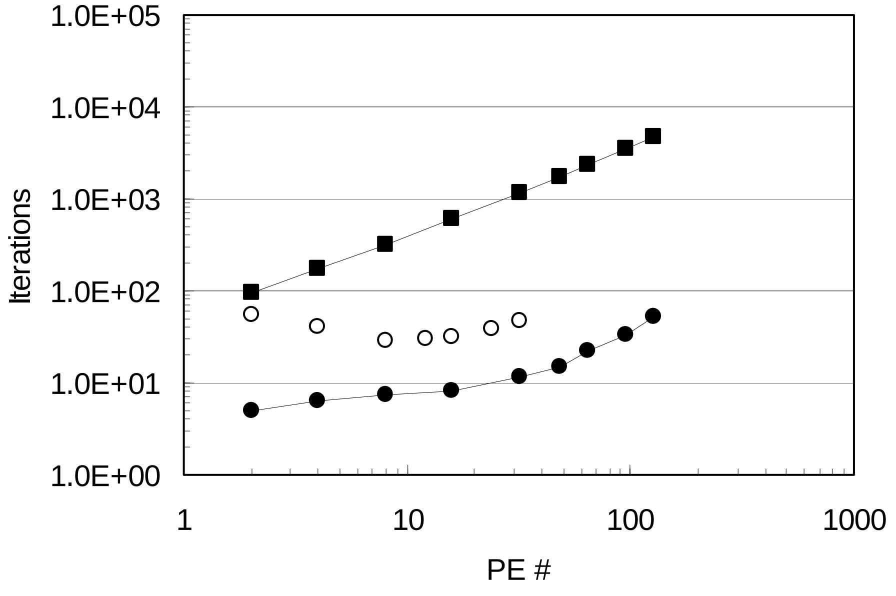


(a) *Uniform* boundary condition

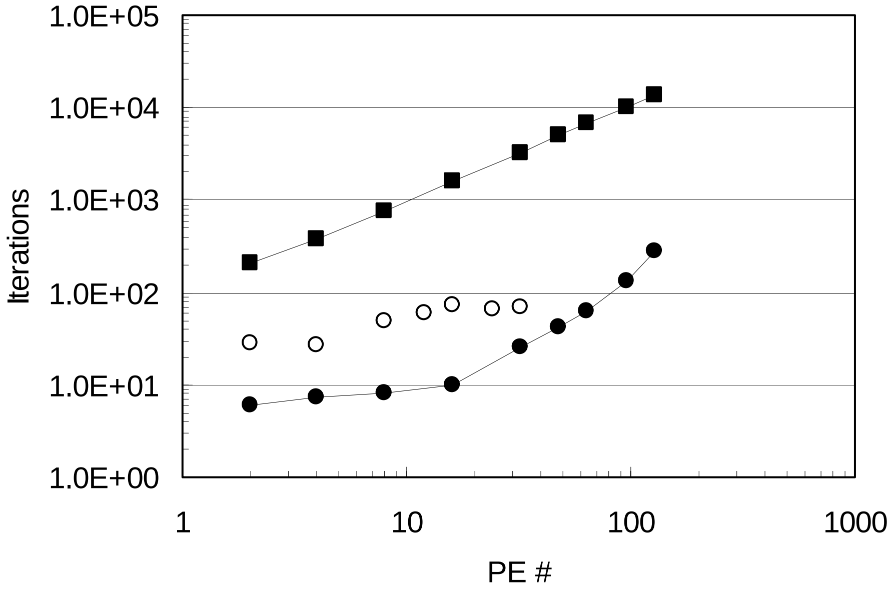


(b) *Single-patch* boundary condition

Fig. 10 Results of application-I. Computation time (including communication for parallel computing) for fixed problem size on each processor (320×900=288,000 cells/PE) for 2 to 32 PEs (up to 9,216,000 cells). Black squares: ICCG, black circles: MGCG/GS, white Circles: MGCG/ILU(0).

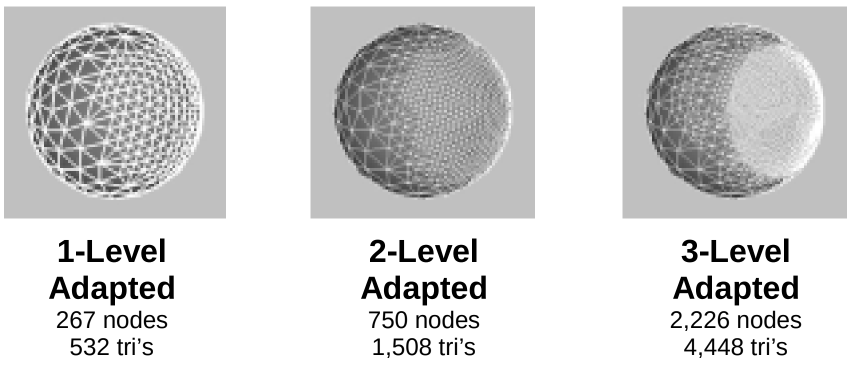


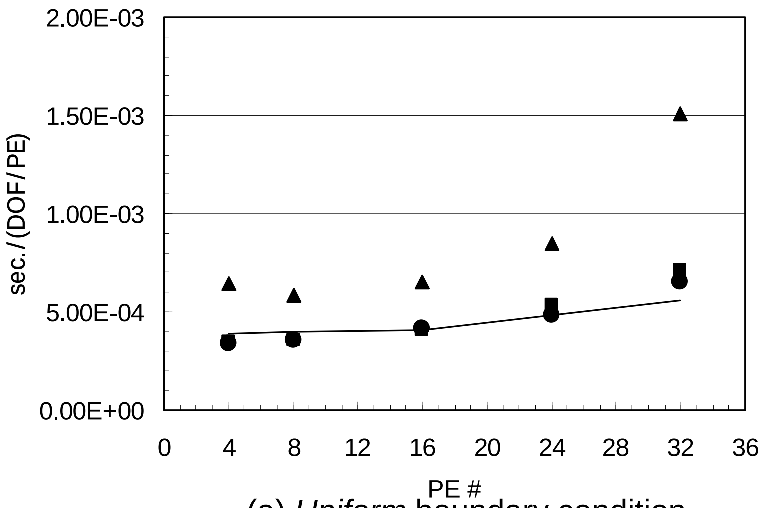
(a) *Uniform* boundary condition



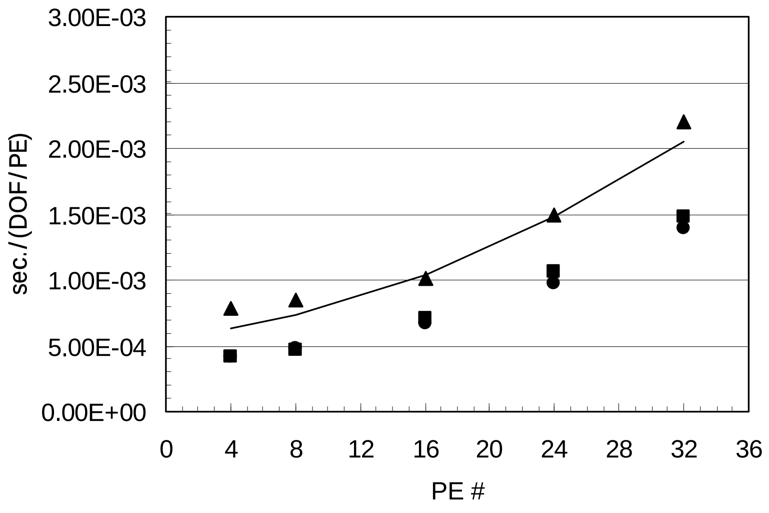
(b) *Single-patch* boundary condition

Fig. 11 Results of application-I. Number of iterations until convergence for fixed problem size on each processor (320×900=288,000 cells/PE) for 2 to 32 PEs (up to 9,216,000 cells). Black squares: ICCG, black circles: MGCG/GS, white circles: MGCG/ILU(0).





(a) *Uniform* boundary condition



(b) *Single-patch* boundary condition

Fig. 12 Results of application-II. Computation time (including communication normalized by cell number/PE for parallel computing) for locally refined grids using MGCG/GS, 50 layers/PE, for 4 to 32 PEs (up to 7,116,800 cells). Solid line: initial grid (320 triangles), circles: 1-level adapted, squares: 2-level adapted, triangles: 3-level adapted.

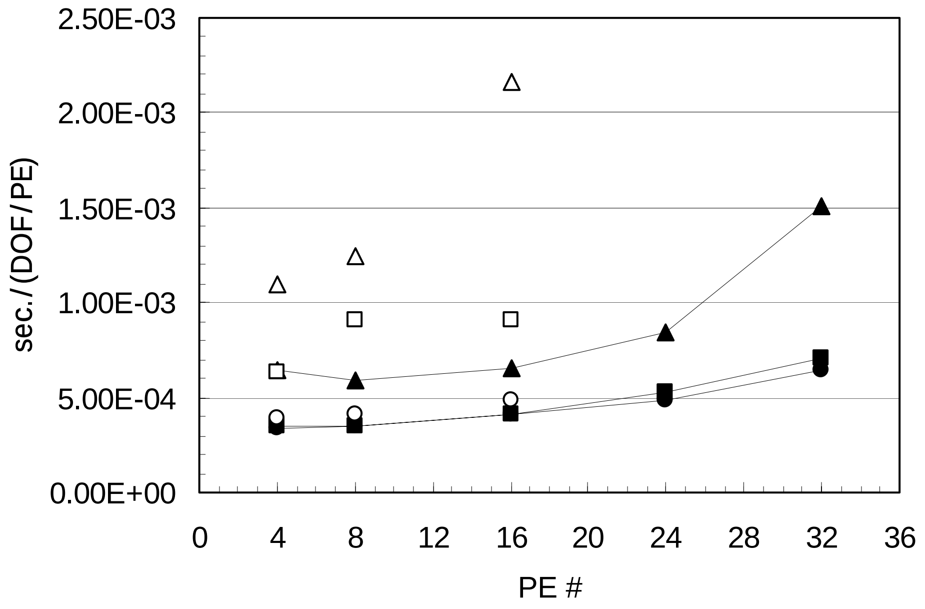


Fig. 13 Results of application-II. Computation time (including communication normalized by cell number/PE for parallel computing) for locally refined grids using MGCG/GS, 50 layers/PE, for 4 to 32 PEs (up to 7,116,800 cells) under the *uniform* boundary condition. Circles: 1-level adapted, squares: 2-level adapted, triangles: 3-level adapted, black symbols: *direct jump method*, white symbols: *level-by-level Method*.

5. Concluding Remarks

A multigrid-preconditioned conjugate gradient iterative method for parallel computers has been developed, in which a V-cycle and semi-coarsening approach is adopted for the multigrid procedure. Both Gauss-Seidel and ILU(0) with additive Schwartz domain decomposition smoothers have been tested. The proposed procedure was applied to Poisson equations in the region between 2 spherical surfaces on adaptively generated semi-unstructured prismatic grids under various boundary conditions. Computational results on a Hitachi SR2201 parallel computer using up to 128 processors demonstrate the good scalability of the method compared with ICCG solvers. Communication overhead is less than 1 % for sufficiently large problems. The efficiency of the Gauss-Seidel smoother becomes worse as the number of PEs increases due to localization, and the ILU(0) smoother is relatively robust for computations across many PEs.

The proposed procedure was also applied to grids with local refinement, and 2 multigrid strategies (*direct jump* and *level-by-level*) were compared. *Direct jump* was found to be much more efficient for deeper-level adaptation despite its simplicity.

Additional robust smoothers will be developed in the future as the next step in this research for application to computations with higher degrees of parallelization (> 1,000 PEs).

Acknowledgments

This study is a part of the *Solid Earth Platform for Large Scale Computation* project funded by the Ministry of Education, Culture, Sports, Science and Technology, Japan through *Special Promoting Funds of Science & Technology*. The author would like to thank Professor Yasumasa Kanada (Computing Center, The University of Tokyo) for helpful discussions on high performance computing.

References

[1] Barrett, R., Bery, M., Chan, T.F., Donato, J., Dongarra, J.J., Eijkhout, V., Pozo, R., Romine, C. and van der Vorst, H. : Templates for the Solution of Linear Systems : Building Blocks for Iterative Methods, SIAM, 1994.
[2] GeoFEM Web Site : http://geofem.tokyo.rist.or.jp/
[3] Briggs, W.L., Henson, V.E. and McCormick, S.F. : A Multigrid Tutorial Second Edition, SIAM, 2000.
[4] Smith, B., Bjϕrstad, P. and Gropp, W. : Domain Decomposition, Parallel Multilevel Methods for Elliptic Partial Differential Equations, Cambridge Press, 1996.
[5] CASC (Center for Applied Scientific Computing, Lawrence Livermore National Laboratory) Web Site : http://www.llnl.gov/CASC/linear_solvers/
[6] Stüben, K. : Algebaic Multigrid (AMG) : An Introduction with Applications, GMD Report 53, GMD-Forschungstentrum Informationstechnik GmbH, 1999.
[7] Adams, M.F. and Demmel, J,W. : "Parallel Multigrid Solver for 3D Unstructured Finite Element Problems ", SC99 Proceedings, Portland, Oregon, USA, 1999.
[8] Parthasarathy, V., Kallinderis, Y. and Nakajima, K. : "A Navier-Stokes Method with Adaptive Hybrid Prismatic/Tetrahedral Grids", AIAA Paper 95-0670, 1995.
[9] Kallinderis, Y. and Nakajima, K. : " Finite Element Method for Incompressible Viscous Flows with Adaptive Hybrid Grids", AIAA Journal, Vol.32, No.8, pp.1617-1625, 1994.
[10] Gropp, W., Lusk, E. and Skjellum, A., : Using MPI, MIT Press, 1994.
[11] The University of Tokyo, Computer Center, Web Site : http://www.cc.u-tokyo.ac.jp/