Parallel Iterative Linear Solvers with Preconditioning for Large Scale Problems

大規模問題のための前処理付き並列反復法

Dissertation

Presented to the Graduate School of
The University of Tokyo
in the Fulfillment of the Requirements
for the Degree of

Doctor of Engineering

Kengo NAKAJIMA, M.S.

中島 研吾

December 2002

Parallel Iterative Solvers with Preconditioning for Large Scale Problems

Kengo Nakajima
The University of Toyko, 2002.

Supervisor: Professor Genki Yagawa, Doctor of Engineering.

Computer simulations are essential for exploring new frontier in science and engineering. Parallel computation is necessary for the numerical solution of various types of large-scale and complicated computation. In many large-scale scientific simulation codes using the finite-element method (FEM) and the finite-difference method (FDM), almost computation is spent in solving linear equations with sparse coefficient matrices. For this reason, much of the scalable algorithm research and development is aimed at solving these large, sparse linear systems of equations on parallel computers.

Among the sparse linear solvers, iterative methods are memory scalable and the only choice for large-scale simulations by massively parallel computers. The rate of convergence of iterative methods depends strongly on the spectrum of the coefficient matrix. Hence, iterative methods usually involve a second matrix that transforms the coefficient matrix into a matrix with more favorable spectrum. The transformation matrix is called a *preconditioner*. The use of a good preconditrioner improves the convergence of the iterative methods.

In this thesis, the following three types of preconditioners of parallel iterative solvers for various types of applications on unstructured meshes were developed using the GeoFEM platform for parallel finite-element methods:

I. **Localized block ILU(0) preconditioning method for 3D solid mechanics on SMP cluster type vector parallel computers, such as the Earth Simulator (general preconditioners).**
II. **Parallel scalable multigrid preconditioning method for 3D Poisson equations derived from incompressible Navier-Stokes solvers with adaptive meshes (preconditioners for broad classes of underlying problems).**

III. Selective blocking preconditioning method for 3D solid mechanics with contact on SMP cluster type vector parallel computers (preconditioners for specific problems).

Adaptive methods in applications with unstructured meshes have evolved as efficient tools for obtaining numerical solution without a priori knowledge of the details of the nature of the underlying physics. However, these methods cause severe load imbalance among processors in parallel computations. In the present study, a parallel mesh adaptation method with dynamic load-balancing using DRAMA library has been developed and implemented on a 3D compressible Navier-Stokes solver developed on the GeoFEM platform. The extended data structure of GeoFEM with mesh adaptation and multigrid procedure has been also proposed.

All of the developed preconditioning methods on the GeoFEM platform for its local data structure provide excellent parallel/vector performance up to > 1,000 PEs and robustness for very ill-conditioned problems. The localized block ICCG(0) solver with special reordering strategy for unstructured mesh attained 3.80 TFLOPS for simple 3D linear elastic problem with 2.2×10^9 DOF on 176 SMP nodes (1,408 PEs) of the Earth Simulator, corresponding to 33.7% of peak performance.

Parallel CG solvers with selective blocking preconditioning and special reordering developed in this study provided excellent performance on the Earth Simulator (29.1% of peak performance) and robustness for ill-conditioned matrices which appear in contact problems. Moreover, selective blocking preconditioning is memory efficient and requires only 25% of ILU(2) and 50% of ILU(1).

The parallel multigrid procedure with new local data structure provided excellent parallel performance of greater than 95% on a Hitachi SR2201 with 128 PEs. The *direct jump* method developed in this study for locally refined mesh is very simple, but was found to be much more efficient than the existing *level-by-level* method described in for deeper-level adaptation. The effect of the parallel multilevel ILU smoother for ill-conditioned problems has been also evaluated.

These methods and data structure are very useful for wide range of scientific applications developed for SMP cluster type architecture which has become very popular for massively parallel computers in recent days.

Acknowledgements

First of all, I would like to thank my advisor Professor Genki Yagawa, University of Tokyo, for devoting a great deal of his time and efforts to this work. His advice always helped me to get a new idea and I have really enjoyed my research work.

Professor Hiroshi Okuda, University of Tokyo, is one of the thesis committee members and always encouraged me to have finished work in recent five years. I greatly appreciate his advice.

I thank Professors Shinobu Yoshimura, Muneo Hori and Shao-Liang Zhang, University of Tokyo for participating the thesis committee and for their valuable suggestions.

This work is performed as a part of GeoFEM project in RIST (Research Organization for Information Science and Technology). I would like to thank all of the current and former members of the GeoFEM developing team: Hisashi Nakamura (RIST), Hiroshi Okuda (University of Tokyo), Issei Fujishiro (Ochanomizu University), Mikio Iizuka (RIST), Kazuteru Garatani (CRC), Hide Sakaguchi (RIST), Mitsuko Hama (RIST), Li Chen (RIST), Hiroaki Matsui (RIST), Jun Yin (RIST), Shin'ich Ezure (RIST), Kazuaki Sakane (RIST), Aya Marumo (RIST), Daigo Sekita (Mitsubishi Research Institute), Yoshitaka Wada (Tokyo University of Science, Suwa), Yuriko Takeshima (Tohoku University), Osamu Hazama (CCSE/JAERI), Norihisa Anan (Yokohama National University), Hiroko Nakamura (Ochanomizu University), Yasuko Suzuki (Mitsubishi Electric), Masaki Nagata (Denso) and Noriyuki Kushida (University of Tokyo).

I would like to thank my academic advisors, Professor Kyohei Kondoh (University of Tokyo, currently National Academy of Defense, Japan) and Professor Yannis Kallinderis (University of Texas at Austin), who opened a door to the world of research for me.

I also would like to thank Professors Yasumasa Kanada (University of Tokyo), Mitsuhiro Matsu'ura (University of Tokyo) and Takahiko Tanahashi (Keio University) for their helpful suggestions.

I started my career as an engineer at the Mitsubishi Research Institute, Inc. in 1985. I had many valuable experiences through various kinds of projects. I would like to thank all of my colleagues there: Hiroshi Saitoh, Kazuhiko Noguchi, Yasushi Kondoh, Hirokazu Tsunoda, Reiji Mezaki, Kumiko Minami, Fumiya Shimizu, Hidenori Yasuda, Takenori Mikasa, Yutaka Kohno, Shingo Ueno, Miyuki Maruyama, Naohiko Nakamura,

Masaaki Matsumoto, Kiyoshi Akizuki (Akizuki Steel), Takashi Imai (ACE Insurance), Hirohisa Noguchi (Keio University), Koichiro Hatanaka (Japan Nuclear Cycle Development Institute), Toshio Nagashima (Sophia University) and Yasuji Fukahori (Beta Engineering).

I am happy to have many friends all over the world through my research works. I really enjoyed interaction with them. I would like to appreciate many suggestions and advice from: Achim Basermann (NEC Europe), Shun Doi (NEC), Jochen Fingberg (NEC Europe), Takashi Furuumura (Earthquake Research Institute, University of Tokyo), Mike Heroux (Sandia National Laboratories), Guy Lonsdale (NEC Europe), Tomoshi Miyamura (Nihon University), Peter Mora (University of Queensland), Esmond Ng (Lawrence Berkeley National Laboratory), Akira Nishida (University of Tokyo), Sachio Ozaki (CRIEPI), Horst Simon (Lawrence Berkeley National Laboratory), Klaus Stüben (SCAI/Fraunhofer), Keita Teranishi (Pennsylvania State University), Takumi Washio (NEC), David Womble (Sandia National Laboratories), Akira Yamaguchi (Japan Nuclear Cycle Development Institute), and Jun Zhang (University of Kentucky).

I also thank all of my family members who have supported me spiritually, especially my parents, Masao and Kazuko.

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*.

Last, but certainly not least, I would like to thank my wonderful wife Reiko for her support throughout this *long and winding road*.

December 2002.
Kengo Nakajima

Contents

Chapter 1 Introduction

In this thesis, parallel iterative solvers with preconditioning for various types of applications on unstructured grids and efficient distributed data structure for parallel computation have been investigated and developed using the GeoFEM platform for parallel finite-element methods. In this chapter, background of this work, especially preconditioned iterative method and parallel programming models are briefly described. The outline of the present work and structure of the thesis are also shown. Finally, features of parallel computer systems used in this thesis are briefly described.

1.1 Why Iterative Methods ?

Computer simulations are essential for exploring new frontier in science and engineering. Parallel computation is necessary for the numerical solution of various types of large-scale and complicated computation. In many large-scale scientific simulation codes using the finite-element method (FEM) and the finite-difference method (FDM), most computation is spent for solving linear equations with sparse coefficient matrices. For this reason, much of the scalable algorithm research and development is aimed at solving these large, sparse linear systems of equations on parallel computers. Sparse linear solvers can be broadly classified as being either *direct* or *iterative*.

Direct solvers such as Gaussian Elimination are based on a factorization of the associated sparse matrix. They are extremely robust and would give the exact solution of $Ax=b$ after a finite number of steps without round-off errors. However, their memory requirements grow as a nonlinear function of the matrix size because original zeroes fill in during factorization.

In contrast, iterative methods are memory scalable. Therefore iterative methods are the only choice for large-scale simulations by massively parallel computers. In [7], iterative methods are classified as being *stationary* or *nonstationary*. Stationary methods are such as the Jacobi method, the Gauss-Seidel method and SOR (Successive Over-relaxation) method. Nonstationary methods are the Krylov subspace methods such as the Conjugate Gradient (CG) method, the Bi-Conjugate Gradient Stabilized (BiCGSTAB) method, Generalized Product-type Bi-Conjugate Gradient (GPBiCG) method and the Generalized Minimal Residual (GMRES) method [7,21,103,125].

Nonstationary methods are usually more complicated but more robust than stationary methods. Nonstationary methods differ from stationary methods in that the computations involve information that changes at each iteration. Typically, constants are computed by taking inner products of residuals, or other vectors arising from the iterative method.

In recent days, various libraries for parallel iterative solvers have been developed and some of them, such as AZTEC [127] and PETSc [142] can be freely downloaded from web-sites.

1.2 Why Preconditioning ?

Iterative methods are memory scalable but their convergence can be slow or they can fail to converge. The rate of convergence of iterative methods depends strongly on the spectrum of the coefficient matrix. Hence, iterative methods usually involve a second matrix that transforms the coefficient matrix into a matrix with more favorable spectrum. The transformation matrix is called a *preconditioner*. The use of a good preconditioner improves the convergence of iterative methods, sufficiently to overcome the extra cost of constructing and applying the preconditioner. Indeed, without a preconditioner the iterative method may even fail to converge.

In the preconditioned iterative methods, original linear equation:

$$Ax = b \tag{1.1}$$

is transformed into the following equation (1.2) using preconditioner M:

$$A'x = b',\ A' = M^{-1}A,\ b' = M^{-1}b \tag{1.2}$$

Equation (1.2) has same solution as (1.1), but the spectral properties of the coefficient matrix $A'=M^{-1}A$ may be more favorable and convergence is faster.

According to [29] preconditioners can be divided roughly into following three categories:

I. Preconditioners designed for wide range of general classes of matrices. Examples of such preconditioners are the Jacobi, Gauss-Seidel, SOR, IC/ILU, and approximate inverse methods. Public libraries such as AZTEC [127] and PETSc [142] usually provide preconditioners in this category.

II. Preconditioners designed for broad classes of underlying problems such as elliptic partial differential equations. Multigrid and domain decomposition preconditioners are classified into this category.

III. Preconditioners designed for a specific matrix or underlying problem. Despite great studies in developing preconditioners for general linear systems or for broad classes of underlying problems, it is still possible in many situations to use physical intuition about a specific problem to develop a more effective preconditioner.

Various types preconditioning methods have been proposed, developed and used. The simplest preconditioning is called diagonal scaling or point Jacobi method where `M` is diagonal components of the original coefficient matrix `A`. Jacobi, Gauss-Seidel and SOR type stationary iterative methods are also well-known as preconditioners. Preconditioning methods using various types of polynomials have been also widely used [7,21,103].

The incomplete lower-upper (ILU) for non-symmetric matrices and incomplete Cholesky (IC) factorization methods for symmetric matrices are the most popular preconditioning techniques for accelerating the convergence of Krylov iterative methods [7,21,103]. ILU/IC methods are based on LU/Cholesky factorization or Gaussian elimination for direct solvers. Procedure of LU factorization or Gaussian elimination is as follows [7,21,103]:

Gaussian Elimination

```
do i= 2, n
  do k= 1,  i-1
    a_jk := a_jk/a_kk
    do j= k+1, n
      a_ij := a_ij - a_ik*a_kj
    enddo
  enddo
enddo
```

In this procedure, many fill-in occurs during factorization, therefore factorized matrix could be dense even if original matrix is sparse [103]. ILU(n) or IC(n) are incomplete factorization where *n*-level fill-in is allowed. Larger *n* provides more accurate factorization and usually leads to robust preconditioning, but more expensive in both memory and CPU time. In many engineering applications, ILU(0)/IC(0) is widely used where there are no fill-in and non-zero pattern of factorized matrix is kept as original coefficient matrix:

ILU(0)

```
do i= 2, n
  do k= 1,  i-1
    if ((i,k) ∈ NonZero(A)) then
      a_jk := a_jk/a_kk
    endif
    do j= k+1, n
      if ((i,j) ∈ NonZero(A)) then
        a_ij := a_ij - a_ik*a_kj
      endif
    enddo
  enddo
enddo
```

ILU(n)/IC(n) with n-level fill-in is described as follows:

ILU(n)

```
LEV_ij=0 if ((i,j) ∈ NonZero(A)) otherwise LEV_ij= p+1
do i= 2, n
  do k= 1,  i-1
    if (LEV_ik ≤ p) then
      a_jk := a_jk/a_kk
    endif
    do j= k+1, n
      if (LEV_ij = min(LEV_ij,1+LEV_ik+ LEV_kj) ≤ p) then
        a_ij := a_ij - a_ik*a_kj
      endif
    enddo
  enddo
enddo
```

ILU(n)/IC(n) ignore numerical values because elements that are dropped depend only on the structure of the original coefficient matrix due to memory requirement. This may cause some difficulties for realistic applications due to memory storage requirement. One of the alternative methods is based on the dropping strategy according to the magnitude of the elements rather than their locations. This type of ILU factorization is generally called ILUT/ICT [82,98,103,115,123,124] where *T* denotes *drop-threshold*.

Approximate inverse method [21,103,115,123] provides a direct approximation to the inverse of original matrix A. This method is useful if the matrix is indefinite and incomplete LU factorization is difficult or impossible. A simple technique for finding approximate inverses of arbitrary sparse matrices is to attempt to find a sparse matrix which minimizes the Frobenius norm of the residual matrix `I-AM`:

$$F(M) = \left\| I - AM^2 \right\|_F \tag{1.3}$$

Domain decomposition method is also a certain category of preconditioning methods. The word "domain decomposition method" covers a wide range of techniques. Basic idea of domain decomposition methods is that entire structure is divided into small pieces (subdomains), problems are solved independently in each subdomain, then results are pieced together in order to give the solution to the entire problem. Domain decomposition methods are divided into two categories, those using *overlapping* subdomains, such as additive and multiplicative Schwartz methods, and those using *nonoverlapping* subdomains, which are sometimes called *substructuring* methods.

For example, additive Schwartz method is considered to be preconditioner:

$$M^{-1} \equiv \sum_{i=1}^{J} B_i, \quad B_i = R_i \left(R_i^T A R_i \right)^{-1} R_i^T \tag{1.4}$$

where R_i is the matrix for permutation index in i-th subdomain and J is the total number of subdomains.

Multigrid is an example of scalable linear solver technology. It uses a relaxation method like Gauss-Seidel to efficiently damp high-frequency error, leaving only low-frequency, or smooth, error. The multigrid idea is to recognize that this low-frequency error can be accurately and efficiently solved for on a coarser (i.e., smaller) grid. Recursive application of this idea to each consecutive system of coarse-grid equations leads to a multigrid V-cycle [11,110]. If the components of the V-cycle are defined properly, the result is a method that uniformly damps all error frequencies with a computational cost that depends only linearly on the problem size. In other words, multigrid algorithms are scalable.

There are two basic multigrid approaches: geometric and algebraic. 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.

In order to enhance multigrid's robustness, it is often used as a preconditioner for Krylov methods such as conjugate gradients. However, since multigrid algorithms tend to be somewhat problem-specific.

1.3 Preconditioned Conjugate Gradient Method

The Conjugate Gradient (CG) method is one of the typical nonstationary method and effective for symmetric positive definite systems. The method proceeds by generating vector sequences of *iterates* (*i.e.,* successive approximations to the solution), residuals corresponding to the iterates, and search directions used in updating the iterates and residuals. Although the length of these sequences can become large, only a small number of vectors needs to be kept in memory. In every iteration of the method, two inner products are performed in order to compute update scalars that are defined to make the sequence satisfy certain orthogonality conditions. On a symmetric positive definite linear system, these conditions imply that the distance to the true solution is minimized in some norm.

The iterates $x^{(i)}$ are updated in each iteration by a multiple (α_i) of the search direction vector $p^{(i)}$:

$$x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)} \tag{1.5}$$

Correspondingly the residuals $r^{(i)} = b - Ax^{(i)}$ are updated as

$$r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)} \quad \text{where } q^{(i)} = Ap^{(i)} \tag{1.6}$$

The choice of $\alpha_i = (r^{(i)T} r^{(i)})/(p^{(i)T} q^{(i)})$ minimizes $r^{(i)T} A^{-1} r^{(i)}$ over all possible choices for a in equation (1.6).

The search directions are updated using the residuals

$$p^{(i)} = r^{(i-1)} + \beta_i p^{(i-1)} \tag{1.7}$$

where the choice $\beta_i = (r^{(i)T} r^{(i)})/(r^{(i-1)T} r^{(i-1)})$ ensures that $p^{(i)}$ and $Ap^{(i-1)}$ (or equivalently, $r^{(i)}$ and $r^{(i-1)}$) are orthogonal. In fact, one can show that this choice of β_i makes $p^{(i)}$ and $r^{(i)}$ orthogonal to all previous $Ap^{(j)}$ and $r^{(j)}$ respectively.

The pseudo code for the preconditioned Conjugate Gradient Method is given in Fig.1.1. It uses a preconditioner M ; for $M=I$ one obtains the unpreconditioned version of the Conjugate Gradient Algorithm. In that case the algorithm may be further

simplified by skipping "**solve**" line, and replacing $z^{(i-1)}$ by $r^{(i-1)}$ (and $z^{(0)}$ by $r^{(0)}$).

The unpreconditioned conjugate gradient method construc the i-th iterate $x^{(i)}$ as an element of $x^{(0)} + \mathrm{span}\{r^{(0)}, Ar^{(0)}, \ldots, A^{i-1}r^{(0)}\}$ so that $(x^{(i)} - \hat{x})^T A(x^{(i)} - \hat{x})$ is minimized where $\hat{x}$ is the exact solution of $Ax=b$. This minimum is guaranteed to exist in general only if A is symmetric positive definite. The preconditioned version of the method uses a different subspace for constructing the iterates, but it satisfies the same minimization property, although over this different subspace. It requires in addition that the preconditioner M is symmetric positive definite.

This minimization of the error is equivalent to the residuals $r^{(i)} = b - Ax^{(i)}$ being M^{-1} orthogonal (that is, $r^{(i)} M^{-1} r^{(j)} = 0$ if $i \neq j$). Since for symmetric A an orthogonal basis for the Krylov subspace $K_i(A, r^{(0)}) \equiv \mathrm{span}\{r^{(0)}, Ar^{(0)}, \ldots, A^{i-1}r^{(0)}\}$ can be constructed with only three-term recurrences, such a recurrence also suffices for generating the residuals. In the Conjugate Gradient method, two coupled two-term recurrences are used; one that updates residuals using a search direction vector, and one updating the search direction with a newly computed residual.

Accurate predictions of the convergence of iterative methods are difficult to make, but useful bounds can often be obtained. For the CG method, the error can be bounded in terms of the spectral condition number κ of the matrix $M^{-1}A$. If E_{max} and E_{min} are the largest and smallest eigenvalues of a symmetric positive definite matrix B, then the spectral condition number of B is $\kappa(B) = E_{max}(B)/E_{min}(B)$. If $\hat{x}$ is the exact solution of the linear system $Ax=b$, with symmetric positive definite matrix A, then for CG with symmetric positive definite preconditioner M, it can be shown that:

$$\left\|x^{(i)} - \hat{x}\right\|_A \leq 2\alpha \left\|x^{(i)} - \hat{x}\right\|_A \quad \text{where } \alpha = \left(\sqrt{\kappa} - 1\right)/\left(\sqrt{\kappa} + 1\right) \qquad (1.8)$$

From this relation, we can see that the number of iterations to reach a relative reduction of ε in the error is proportional to $\sqrt{\kappa}$.

In some cases, practical application of this error bound is straightforward. For example. Elliptic second order partial differential equations typically give rise to coefficient matrices A with $\kappa(A)=O(h^{-2})$ (where h is the discretization mesh width),

independent of the order of the finite elements or differences used, and of the number of space dimensions of the problem. Thus, without preconditioning, we can expect a number of iterations proportional to h^{-1} for the CG method.

1.4 Parallel Programming Models

1.4.1 Overview

Recently, different parallel architectures have emerged, each with its own set of programming paradigms. These are classified into, *message passing*, *shared-memory directives* and *hybrids* of the message passing and shared-memory directives. A brief description of each model will be given in this section.

1.4.2 Message Passing

Parallel programming with message passing is the most common and mature approach for parallel systems. On distributed memory architectures, each processor has its own local memory that only it can access directly. In order to access the memory of another processor, a copy of the desired data must be explicitly sent across the network using a message-passing library such as MPI [139]. To run a code on such machines, the programmer must decide how the data should be distributed and communicated among the processors. This model requires a complex program structure, especially for irregularly structured applications. But performance for coarse-grained communication and implicit synchronization through blocking communication provide benefits.

1.4.3 Shared Memory

Using a shared-memory system can greatly reduce the number of programming tasks compared to message-passing paradigm. In distributed shared-memory architectures, each processor has a local memory as well as direct access to all of the memory in the system. Parallel programs are relatively easy to implement since each processor has a global view of the entire memory. Parallelism can be achieved by inserting compiler directives into the code in order to distribute loop iterations among the processors. However, performance may suffer from poor spatial locality if the parallelism is not properly configured.

OpenMP [14,141] is a specification for a set of compiler directives, library routines, and environment variables that can be used to specify shared memory parallelism in Fortran and C/C++ programs. OpenMP is designed for Fortran, C and C++ to support the language that the underlying compiler supports. The fine-grain parallelism is expressed by loop-level compiler directives. The syntax is similar to the native pragma directives of many vendors.

1.4.4 SMP Cluster Architecture and Hybrid Parallel Programming Model

Recent technological advances have allowed increasing numbers of processors to have access to a single memory space in a cost-effective manner. As a result, symmetric multiprocessor (SMP) cluster architectures have become very popular as teraflop-scale parallel computers, such as the Accelerated Strategic Computing Initiative (ASCI) [126] machines and the Earth Simulator [130].

In order to achieve minimal parallelization overhead, a multi-level *hybrid* programming model [13,20,23,86] is often employed for SMP cluster architectures (Fig.1.2). The goal of this method is to combine coarse-grain and fine-grain parallelism. Coarse-grain parallelism is achieved through domain decomposition by message passing among SMP nodes using a scheme such as Message Passing Interface (MPI) [139], and fine-grain parallelism is obtained by loop-level parallelism inside each SMP node by compiler-based thread parallelization such as OpenMP [14,141].

Another commonly used programming model is the single-level *flat MPI* model [13,20,23,86] (Fig.1.2), in which separate single-threaded MPI processes are executed on each processing element (PE). The advantage of a hybrid programming model over flat MPI is that there is no message-passing overhead in each SMP node. This is achieved by allowing each thread to directly access data provided by other threads by accessing the shared memory rather than by message passing. However, a hybrid approach usually requires more complex programming.

Although a significant amount of research on this issue has been conducted in recent years [13,23], most studies have focused on applications involving structured grids such as the NAS Parallel Benchmarks (NPB) [140], with very few examples treating unstructured grids [20,81,86]. Moreover, it remains unclear whether the performance gains of this hybrid approach compensate for the increased programming complexity. Several examples indicate that *flat MPI* is somewhat better, although the efficiency depends on hardware performance (CPU speed, communication bandwidth, and memory bandwidth), features of applications, and problem size [97].

1.5 GeoFEM Project

The Ministry of Education, Culture, Sports, Science and Technology (MEXT), Japan has begun an Earth Simulator project from the fiscal year of 1997 for predicting various Earth phenomena through the simulation of virtual Earth placed in a supercomputer, called "Earth Simulator (40 Tflops/peak)". The Earth Simulator has shared memory symmetric multiprocessor (SMP) cluster architecture and consists of 640 SMP nodes, where each SMP node consists of 8 vector processors [130]. GeoFEM [131] was developed as a parallel finite-element platform for solid earth simulation on the Earth Simulator.

GeoFEM challenges for long-term prediction of the activities of the plate and mantle near the Japanese islands through the modeling and calculation of the solid earth analysis, including the dynamics and heat transfer inside the Earth. GeoFEM is composed of "analysis modules" for structural/electromagnetic thermal fluid / wave propagation simulations and "platform" for parallel I/O / equation solvers / visualization functions, as shown in Fig.1.3. System is designed to be pluggable such that each analysis module is replaceable and that communications among PEs are done implicitly in platform. Platform includes a "coupler", supporting communications among analysis modules for multi-disciplinary computations. "Utilities" i.e. mesh partitioners and pre/post viewer are also supplied. ICCG solver tuned for SMP parallel / vector hybrid system is expected to attain over 10 Tflops on the Earth Simulator. In collaboration with solid earth researchers, various modules are being developed in such fields as earthquake cycle, faulting and propagation, mantle/core dynamics, seismic wave propagation, GPS data assimilation etc.

In GeoFEM, preconditioned iterative method is implemented in order to solve large-scale problems with more than 10^8 DOF. GeoFEM solves both of symmetric and un-symmetric matrices. Therefore CG (Conjugate Gradient) for symmetric matrices and BiCGSTAB (Bi-Conjugate Gradient Stabilized) GPBiCG (Generalized Product-type Bi-Conjugate Gradient) and GMRES (Generalized Minimal Residual) methods are implemented [7,21,103,125]. GMRES is especially suitable for nonlinear problems. Some of the applications in GeoFEM consist of highly nonlinear process such as contact of fault zones for earthquake generation simulation. Usually, this type of application provides very ill-conditioned matrices for linear equations and these are very difficult to solve using iterative methods. In GeoFEM, various types of robust preconditioning methods were developd. Some of them are general and applicable for wide range of

applications (*e.g.* ILU(1), ILU(2) etc.). Problem specific preconditioning methods (*e.g. selective blocking* for contact problems) have been also developed.

1.6 Present Work

In the present work, parallel iterative solvers with preconditioning for various types of applications on unstructured grids and efficient distributed data structure for parallel computation have been investigated and developed using the GeoFEM platform for parallel finite-element methods.

The following three types of preconditioners for nonstationary (or Krylov subspace) iterative solvers were developed each of which corresponds to each of the three categories of preconditioners in Section 1.2 on the GeoFEM platform:

I. **Localized block ILU(0) preconditioning method for 3D solid mechanics on SMP cluster type vector parallel computers, such as the Earth Simulator**: 3-level hybrid parallel programming model with inter/intra-SMP node communication and vectorization on individual PE has been developed. This is a very general preconditioner which can be applied to various types of applications.

II. **Parallel scalable multigrid preconditioning method for 3D Poisson equations derived from incompressible Navier-Stokes solvers with adaptive meshes**: Extended data structure for multilevel method has been developed. Various smoothers for multigrid procedure have been evaluated under various types of boundary conditions. This is a preconditioner for broad class of underlying problems.

III. **Selective blocking preconditioning method for 3D solid mechanics with contact on SMP cluster type vector parallel computers**: This method was developed for contact simulations of earthquake generation at fault zones. This is a problem specific preconditioner for very ill-conditioned applications.

In this work, symmetric positive definite matrices which satisfy $(Au, u) > 0, \forall u \in \mathbf{R}^n, u \neq 0$ are mainly treated which are derived from descritization of certain problems such as Poisson equations, linear elastic equations for solid mechanics and linearized Newton-Raphson equations for contact problems

without friction. But the idea in this paper can be extended to other types of problems such as equations with un-symmetric matrices. Therefore, the Conjugate Gradient (CG) method is mainly used as iterative method [7]. Message passing type programming model is adopted and the program is written in Fortran 90 with MPI [139].

In I and III, preconditioning methods are optimized for SMP cluster architectures with vector processors using *hybrid* programming model [75,77,79,81].

As for the multigrid method in II, geometrical multigrid approach [74,78,80] utilizing hierarchical data structure for adaptively generated meshes and local mesh refinement were developed. Recently, many research on parallel mutigrid method has been conducted and some of them are for problems with local mesh refinement for adaptation. But most of the studies on parallel multigrid methods with mesh adaptation have focused on block-structure type grids and there have been very few works on unstructured meshes. In this work, a new algorithm for multigird procedure with local mesh refinement has been developed.

Adaptive methods in applications with unstructured meshes have evolved as efficient tools for obtaining numerical solution without a priori knowledge of the details of the nature of the underlying physics. But these methods cause severe load imbalance among processors in parallel computations. In the present study, parallel mesh adaptation method with dynamic load-balancing using DRAMA library [129] has been developed and implemented into 3D compressible Navier-Stokes solver developed on the GeoFEM platform. Extended data structure of GeoFEM with mesh adaptation has been also proposed.

1.7 Overview of Thesis

The thesis presents and evaluates parallel iterative solvers with preconditioning for various types of applications on unstructured grids and an efficient distributed data structure for parallel computation that have been investigated and developed using the GeoFEM platform for parallel finite-element methods.

Chapter 2 presents an overview of GeoFEM's distributed data structure and parallel iterative solvers for unstructured grids [71,72,73,79,81,84,131].

Chapter 3 describes parallel iterative solvers using the localized block ILU(0)/IC(0) preconditioning method for SMP cluster type vector parallel computers and the results of 3D elastic problems with 2. 2 G DOF using 176 SMP nodes of the Earth Simulator are presented [79,81]. The performance of the *flat MPI* and the *hybrid* parallel programming model described in Section 1.4 is evaluated.

In Chapter 4, the selective-blocking preconditioning method for contact problems is described and large-scale problems are solved using a Hitachi SR2201 with a flat MPI and the Earth Simulator with a hybrid parallel programming model.

Chapter 5 presents the parallel multigrid preconditioning method for locally refined unstructured meshes [74,78,80]. The developed method is evaluated on a Hitachi SR2201 using a flat MPI parallel programming model.

Chapter 6 shows the parallel mesh adaptation procedure with dynamic load balancing on the GeoFEM platform. The extended distributed data structure is also proposed [76]. Flat MPI is implemented on a PC cluster and a Hitachi SR2201.

Finally, Chapter 7 presents the main conclusions of this thesis, as well as recommendations for future work.

1.8 Environments for Parallel Computation

1.8.1 Overview

In the present study, following computers were used for serial and parallel computation:

- COMPAQ Alpha Cluster with Alpha21164 500/599 MHz, 16 PEs at RIST [131]
- LAMP Cluster with Pentium-Pro 200 MHz, 32 PEs at NEC Europe [137]
- Hitachi SR2201 with 1024 PEs at University of Tokyo [132]
- Hitachi SR8000/128 with 128 SMP nodes (1024 PEs) at University of Tokyo [133]
- Earth Simulator with 640 SMP nodes (5120 PEs) [130]

In the following part of this subsection, features of Hitachi SR2201, Hitachi SR8000/128 and the Earth Simulator will be briefly described.

1.8.2 Hitachi SR2201

In this study, a Hitachi SR2201 system at the Information Technology Center of the University of Tokyo was employed. The system has 1,024 PEs, where each PE has 300 MFLOPS peak performance and 256 MB memory. The total system provides 300 GFLOPS of peak performance and 256 GB memory [135]. Hardware of a Hitachi SR2201 system has the following features:

- Pseudo-vector processing function
- Three-dimensional crossbar switch network
- Partitioned operation

(1) Pseudo-vector processing function

High-speed numerical computations in the microprocessor are achieved by *pseudo-vectorization* [132]. Each microprocessor in a node pipelines data from memory without interrupting subsequent instructions. Therefore, high-speed large-scale computing is possible by supplying a large amount of data to the computing element from memory.

Generally, a RISC microprocessor-based machine has a cache memory between the processor and the main memory for high-speed data transmission to the processor,

thereby increasing performance. For numerical calculation programs such as FORTRAN, however, the cache memory cannot be fully utilized because a large range of array data is defined and referenced through loops, eventually lowering performance. As a solution to this performance reduction, the SR2201 provides pseudo-vector processing for high-speed transmission of data from the memory to the processor. Pseudo-vector processing generates an object program that processes the data referenced in a loop in one of the following ways.

- The data is loaded in advance in a floating-point register, and loading is completed while the loop that references the data is performing calculations from previous iterations. (preload optimizing)
- The data is transferred is advance into a memory cache, and the transfer is completed while the loop that references the data is performing calculations from previous iterations. (prefetch optimizing)

(2) Three-dimensional crossbar switch network

One of the key architectural considerations in ensuring an excellent cost/performance ratio is the reduction of overhead associated with pre- and post-processing. Three-dimensional crossbar switch of Hitachi SR2201 provides high-speed connection among individual processing elements (PEs). With this switch, there are only three output lines from any PE: one for each of the crossbars. This simple layout achieves almost the same performance as the configuration which interconnects all the processing elements directly, yet at a much lower cost. Features of the crossbar switch are as follows:

- A crossbar switch consists of 3 crossbars, one for each axis (X, Y & Z), to create a 2-dimensional or a 3-dimensional structure.
- An X-crossbar switch is capable of switching up to 8 x 8 connections. A Y-crossbar switch and a Z-crossbar switch are capable of switching up to 16 x 16 connections.
- Data transfer rate: 300MB/s in each direction of the dual ports.

(3) Partitioned operation

To ensure flexibility in operation, the SR2201 series supports partitioned operation. The entire system can be partitioned into a maximum of 8 groups (partitions), each of which

can execute its own job stream independently of the others.

1.8.3 Hitachi SR8000/128

The Hitachi SR8000 is a distributed-memory parallel system with 4 to 128 configurable nodes. The nodes are connected by a high-speed multidimensional crossbar network and each node consists of multiple (8) microprocessors (IPs). These IPs perform high-speed operation simultaneously via the cooperative microprocessor (COMPAS) feature [133].

In this study, a Hitachi SR8000/128 system at the Information Technology Center of the University of Tokyo was employed. The system has 128 SMP nodes, where each node has 8 PEs, 8GFLOPS peak performance and 8GB memory. The total system provides 1.0 TFLOPS of peak performance and 1.0 TB memory [135].

Cooperative microprocessors (COMPAS) [133] provides high-speed simultaneous activation of multiple processors in a node. Each microprocessor in the node executes one of the threads into which the original program is divided. The compiler automatically performs parallelization in the node, allowing the user to code data without being aware of hardware architecture. Parallelization of vector operations simplifies conversion from the standard vector operations.

Pseudo-vectorization is also available in SR8000 for high-speed numerical computations.

1.8.4 Earth Simulator

The Earth Simulator is based on:

- 5,120 (640×8-way nodes) 500 MHz NEC CPUs
- 8 GFLOPS per CPU (40 TFLOPS total)
- 2 GB (4×512 MB FPLRAM modules) per CPU (10 TB total)
- 640×640 crossbar switch between the nodes
- 16 GB/s inter-node bandwidth
- 20 kVA power consumption per node

The vector CPU is made using 0.15 μm CMOS process, and is a descendant (same speed, smaller process) of the NEC SX-5 CPU. The machine runs a version of the Super-UX UNIX-based OS. OpenMP parallel directives are used within a node, and MPI-2 or HPF must be used across multiple nodes, necessitating a dual-level parallel implementation. In fact this can be considered a three-level parallel system, if single-CPU vectorization is taken into account; however, vectorization is largely

automatic. Still, an optimized code will need to employ MPI-2 at the subdomain level, OpenMP at the loop level, and vectorization directives at the instruction level all at once.

The CPUs are housed in 320 cabinets, 2×8-CPU nodes per cabinet. Figure 1.4 shows that the cabinets (purple) are organized in a ring around the interconnect, which is housed in another 65 cabinets (blue). Another layer of the circle is formed by disk array cabinets (white), as shown in Fig.1.4. The whole thing occupies a building 65 m long and 50 m wide. Activity on the nodes is signaled by a green beacon at the top of the cabinet, and if a fault occurs, a similar red light turns on. Switch cabinets also have green and red signaling lights for various types of communication events.

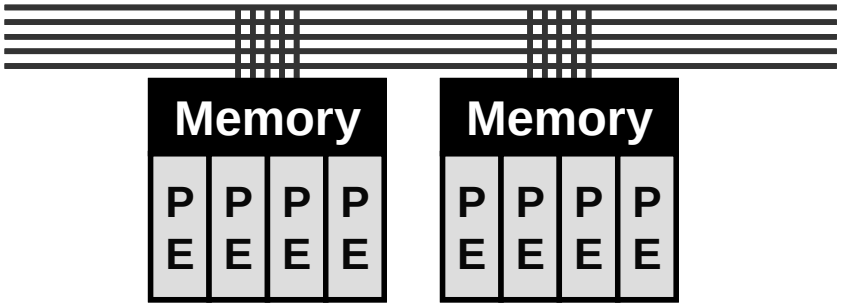
1.8.5 Parallel Computers in this Thesis

Following table describes the computers and parallel programming models used in each chapter of this thesis, with information for the category of the preconditioning methods.

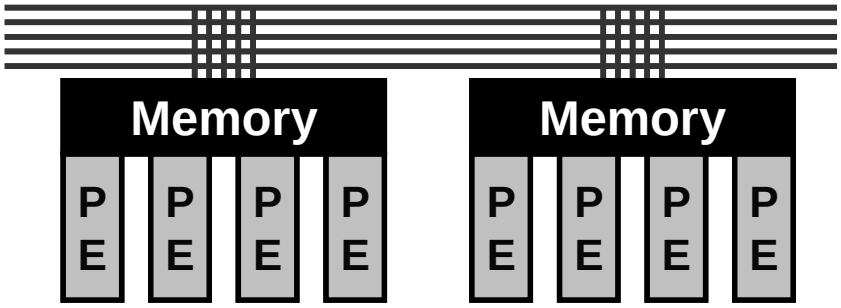
	Category of Precondi-tioner	Alpha Cluster	LAMP (Pentium-Pro) Cluster	Hitachi SR2201	Hitachi SR8000	Earth Simulator
Programming Model	—	Serial	Flat MPI	Flat MPI	Hybrid	Flat MPI Hybrid
Chap.2: **Test for parallel iterative solvers in GeoFEM**	I			○		
Chap.3: **Localized BICCG(0) solvers for SMP cluster architectures**	I				○	○ Flat MPI Hybrid
Chap.4: **Selective blocking preconditioning for contact problems**	III	○		○		○ Hybrid only
Chap.5: **Parallel multigrid preconditioning for Poisson equations**	II			○		
Chap.6: **Adaptive mesh refinement and dynamic load-balancing**	—		○	○		

```
compute r(0)= b – Ax(0) for some initial guess x(0)
for i= 1,2,...
   solve M z(i-1)= r(i-1) (M: preconditioning matrix)
   ρi-1= r(i-1)T z(i-1)
   if i=1
     p(1)= z(0)
    else
     βi-1= ρi-1/ρi-2
     p(i)= z(i-1) + βi-1 p(i-1)
   endif
   q(i)= A p(i)
   αi= ρi-1/(p(i)T q(i))
   x(i)= x(i-1) + αi p(i)
   r(i)= r(i-1) - αi q(i)
   check convergence; continue if necessary
end
```

Fig. 1.1 Procedure of Conjugate Gradient Method (CG) [7,21]



Hybrid : Hierarchy



Flat-MPI : Each PE -> Independent

	Each PE	Intra NODE	Inter NODE
Hybrid	F90 + directives (OpenMP)		MPI
Flat-MPI	F90	MPI	

Fig. 1.2 Parallel programming models for SMP cluster architectures [3,4,5,6,10]

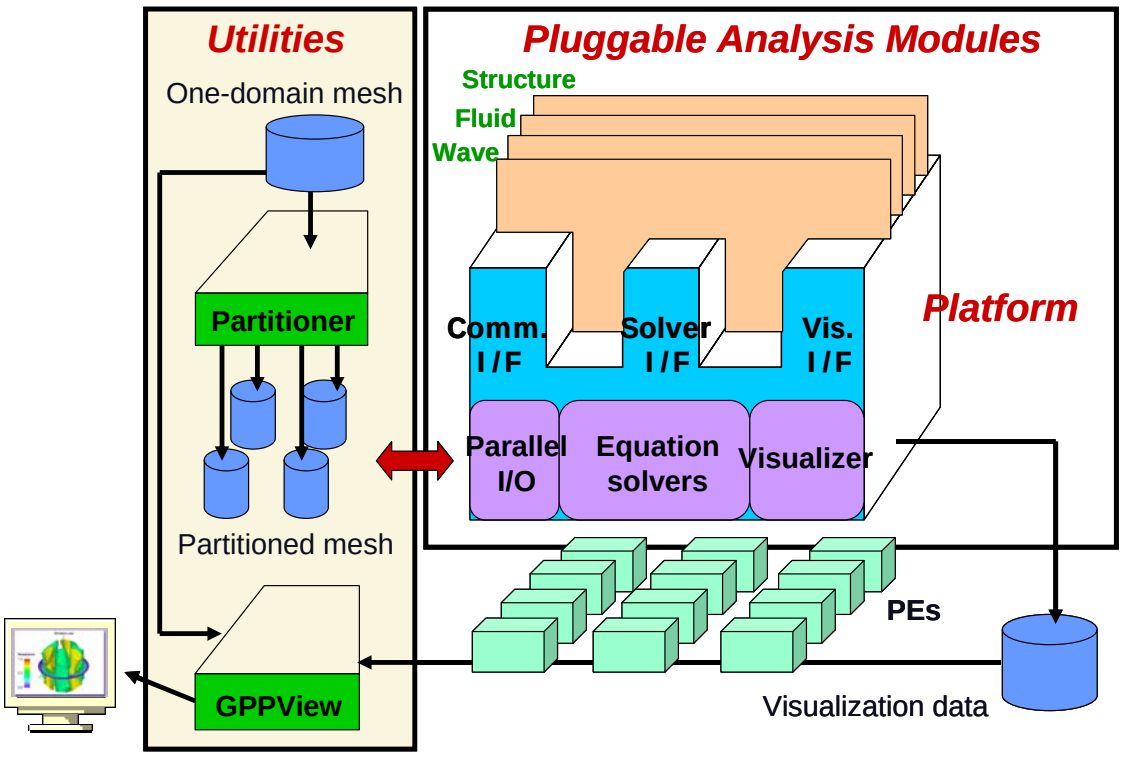


Fig. 1.3 System Configuration of GeoFEM [84,131]

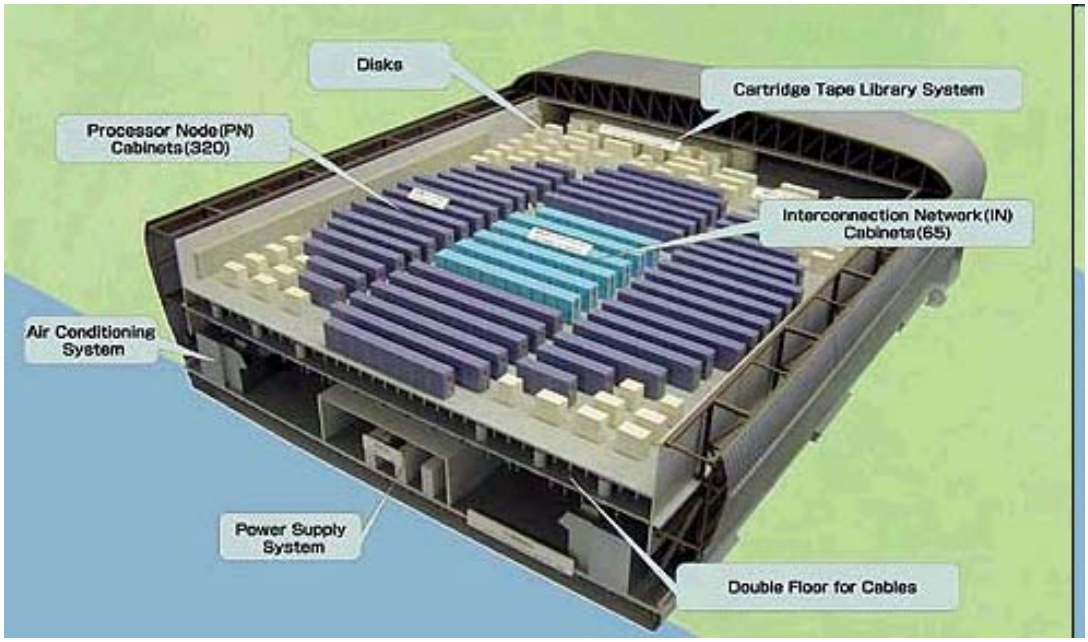


Fig. 1.4 Earth Simulator [130]

Chapter 2 Parallel Iterative Solvers in GeoFEM

In many large-scale scientific simulation codes using finite-element method (FEM) and finite-difference method (FDM), almost computation is spent in a linear solver. For this reason, much of the scalable algorithm research and development is aimed at solving these large, sparse linear systems of equations on parallel computers.

In GeoFEM, preconditioned iterative methods have been adopted. A proper definition of the layout of the distributed data structures is very important for the efficiency of parallel computations with unstructured meshes.

In this chapter, outline of the local data structure and parallel iterative methods with localized preconditioning in GeoFEM are described.

2.1 Procedure of Parallel FEM

In the simulations by finite-element method (FEM), users usually do computations according to the following procedures (Fig.2.1):

- Pre-Processing (*e.g.* mesh generation etc.)
- Main Simulation (*e.g.* structural analysis, fluid analysis etc.)
- Post-Processing (*e.g.* visualization, data mining etc.)

In parallel computation, data size is potentially very large. Therefore, the entire data set should be partitioned into small local data sets in order to perform efficient local operations. Each domain (or partition) is assigned to each PE (processing element) of the parallel computers. This type of parallel processing is very suitable for FEM due to its locality of the procedures (Fig.2.2). Most of the computation time in FEM procedure is spent for the following two processes:

- Assembling for coefficient matrix
- Solving linear equations

Matrix assembling part is conducted by purely *element-by-element* manner, therefore this process can be parallelized perfectly and no communication among processors occurs during this process. Communication occurs only in the linear solvers. Thus, entire code developed for PC's with single processor can be ported to parallel computers easily except linear solvers.

2.2 Distributed Data Structure

A proper definition of the layout of the distributed data structures is an important factor determining the efficiency of parallel computations with unstructured meshes. The local data structures in GeoFEM are node-based with overlapping elements, and as such are appropriate for the preconditioned iterative solvers used in GeoFEM [131].

In FEM, independent variables for linear equations (*e.g.*, velocity, temperature etc.) are defined on nodes. From the viewpoint of efficiency in parallel computation, the number of the nodes should be balanced among the domains. Therefore, GeoFEM adopts node-based partitioning method. In node-based manner for partitioning, overlapping elements among the domains are required in order to *element-by-element* operations in FEM procedure such as matrix assembling. Fig.2.3 shows the example of overlapping elements. Each node requires information from all of the elements surrounding the node. In Fig.2.3, elements in gray color is shared by more than two domains and information of these elements are required in order to complete the process for each node. Each domain must consist of information of overlapping elements in order to conduct *element-by-element* procedure in purely parallel manner.

Communication among processors occurs during computation. Subroutines for communications in structured grids are provided by MPI. However, users are required to design both the local data structure and communications for unstructured grids. In GeoFEM, each domain contains the following local data:

- Nodes originally assigned to the domain
- Elements that include the assigned nodes
- All nodes that form elements but are from external domains
- A communication table for sending and receiving data
- Boundary conditions and material properties

Nodes are classified into the following 3 categories from the viewpoint of message passing:

- Internal nodes (originally assigned to the domain)
- External nodes (forming the element in the domain but are from external domains)
- Boundary nodes (external nodes of other domains)

Fig.2.4 shows the sample partitioning. If the PE #2 partition in Fig.2.4 and Fig.2.5 is considered, nodes are classified as follows:

- Internal nodes {1,2,3,4,5,6}
- External nodes {7,8,9,10,11,12}
- Boundary nodes {1,2,5,6}

Communication tables between neighboring domains are also included in the local data. Values on *boundary* nodes in the domains are *sent* to the neighboring domains and are *received* as *external* nodes at the *destination* domain. This data structure, described in Fig.2.4, and communication procedure described in Fig.2.5 provide excellent parallel efficiency [28,71,72,73,79,81]. Fig.2.6 describes Fortran subroutine of communication procedures in GeoFEM. In this figure, the arrays `EXPORT_INDEX` and `EXPORT_NODE` correspond to communication table for send-phase and `IMPORT_INDEX` and `IMPORT_NODE` correspond to receiving part of the communication table.

This type of communication occurs in the procedure for computing the matrix-vector product of Krylov iterative solvers described in the next subsection. The partitioning program in GeoFEM works on a single PE, and divides the initial entire mesh into distributed local data.

2.3 Localized Preconditioning

The incomplete lower-upper (ILU) and incomplete Cholesky (IC) factorization methods are the most popular preconditioning techniques for accelerating the convergence of Krylov iterative methods.

Of the range of ILU preconditioning methods, ILU(0), which does not allow fill-in beyond the original non-zero pattern, is the most commonly used. Forward/backward substitution (FBS) is repeated at each iteration. FBS requires global data dependency, and this type of operation is not suitable for parallel processing in which locality is of utmost importance. Most preconditioned iterative processes are a combination of the following operations:

- Matrix-vector products
- Inner dot products
- DAXPY ($\alpha\mathbf{x}+\mathbf{y}$) operations [21] and vector scaling
- Preconditioning operations

Figure 2.7 shows procedures of preconditioned CG iterative method [7,21]. According to this figure, preconditioned CG process consists of:

- Matrix-vector products 1
- Inner dot products 2
- DAXPY 3
- Preconditioning operations 1

per each iteration cycle. The first 3 operations can be parallelized relatively easily. Fig.2.8 shows these 3 procedures parallelized by FORTRAN with MPI. In *matrix-vector products*, communication table defined in the previous subsection is utilized. `MPI_ALLREDUCE` subroutine provided by MPI is useful for *inner dot products*. No communication is required for *DAXPY* procedure.

In general, preconditioning operations such as FBS represent almost 50 % of the total computation if ILU(0) is implemented as the preconditioning method. Therefore, a high degree of parallelization is essential for the FBS operation. But it is well-known that FBS process is difficult to parallelize due to *global* data dependency, as shown in the following equations:

Forward Substitution $$y_k = b_k - \sum_{j=1}^{k-1} L_{kj}\, y_j \quad (k = 2, \cdots N)$$

Backward Substitution $$x_k = \tilde{D}_k \left(b_k - \sum_{j=k+1}^{N} U_{kj}\, y_j \right) \quad (k = N, N-1, \cdots 1)$$

The localized ILU(0) used in GeoFEM is a *pseudo* ILU(0) preconditioning method that is suitable for parallel processors. This method is not a *global* method, rather, it is a *local* method on each processor or domain. The ILU(0) operation is performed locally for a coefficient matrix assembled on each processor by zeroing out components located outside the processor domain (Fig.2.9). This is equivalent to solving the problem within each processor with zero Dirichlet boundary conditions during the preconditioning. This *localized* ILU(0) provides data locality on each processor and good parallelization because no inter-processor communications occur during ILU(0) operation. This idea is originally from the incomplete block Jacobi preconditioning method [21,107]. Fig.2.10 shows the parallel CG iterative method using localized preconditioning. Communication occurs only three times per each iteration cycle, where two of them are only broadcasting of the scalar and one is for information exchange in overlapped region using communication table.

However, localized ILU(0) is not as powerful as the global preconditioning method. Generally, the convergence rate degrades as the number of processors and domains increases [28,71,72,73]. At the critical end, if the number of processors is equal to the number of degrees of freedom (DOF), this method performs identically to diagonal scaling. Table 2.1 shows the results of a homogeneous solid mechanics example with 3×44^3 DOF solved by the conjugate gradient (CG) method with localized IC(0) preconditioning. Computations were performed on the Hitachi SR2201 in the University of Tokyo. Although the number of iterations for convergence increases according to the domain number, this increase is just 30% from one to 32 PEs.

Evaluations for parallel performance were conducted also on the Hitachi SR2201 at the University of Tokyo. Figure 2.11 shows the work ratio (real computation time/elapsed execution time including communication) for various problem sizes [79,81] of simple 3D elastic problems with homogeneous boundary conditions. In these computations, the problem size for 1 PE was fixed. The largest case was 196,608,000 DOF on 1,024 PEs. Figure 2.11 shows that the work ratio is higher than 95% if the

problem size for 1 PE is sufficiently large. In this case, code was vectorized and a performance of 68.7 GFLOPS was achieved using 1,024 PEs. Peak performance of the system was 300 GFLOPS with 1,024 PEs; 68.7 GFLOPS corresponds to 22.9% of the peak performance [79,81]. This good parallel performance is attributed largely to the reduced overhead provided by the use of communication tables as part of GeoFEM's local data structure.

Proper ordering and partitioning provides perfect *global* ILU/IC preconditioning [103], but this is possible only when entire matrix has been obtained. Therefore this is not applied to the present study where coefficient matrices are generated in each domain. In [35], parallel global ILU(k) preconditioing procedure was developed. Robust convergence has been provided but parallel efficiency was not good.

2.4 Additive Schwarz Domain Decomposition

In order to stabilize localized ILU(0) preconditioning, additive Schwarz domain decomposition (ASDD) for overlapped regions [107] has been introduced. The procedure is as follows :

（1） Global preconditioning $Mz = r$ is performed where M is a preconditioning matrix and r and z are vectors.

（2） If the entire domain is divided into 2 domains Ω_1 and Ω_2 , such as in Fig.2.12(a), the preconditioning matrix is solved locally via localized preconditioning according to :

$$z_{\Omega_1} = M_{\Omega_1}^{-1} r_{\Omega_1}, \quad z_{\Omega_2} = M_{\Omega_2}^{-1} r_{\Omega_2}$$

（3） After the local preconditioned matrices are solved, the effects of overlapping regions Γ_1 and Γ_2 are introduced by the following global nesting correction (Fig.2.12(b)):

$$z_{\Omega_1}^{n} = z_{\Omega_1}^{n-1} + M_{\Omega_1}^{-1}(r_{\Omega_1} - M_{\Omega_1} z_{\Omega_1}^{n-1} - M_{\Gamma_1} z_{\Gamma_1}^{n-1})$$

$$z_{\Omega_2}^{n} = z_{\Omega_2}^{n-1} + M_{\Omega_2}^{-1}(r_{\Omega_2} - M_{\Omega_2} z_{\Omega_2}^{n-1} - M_{\Gamma_2} z_{\Gamma_2}^{n-1})$$

where n denotes the number of cycles of the additive Schwarz domain decomposition.

（4） Repeat steps (2) and (3) until convergence

Table 2.2 shows the effect of ASDD for a solid mechanics example with 3×44^3 DOF. Computations were performed on a Hitachi SR2201 at the University of Tokyo with 1 ASDD cycle per iteration. Without ASDD, the number of iterations for convergence increases according to the number of partitions. In contrast, when ASDD is introduced, the number of iterations until convergence remains constant, although the computation time for a single iteration increases.

2.5 Summary

In this chapter, outline of local data structure and parallel iterative solvers with localized preconditioning in GeoFEM was described. Well-designed local data structure with communication tables and localized preconditioning method provide highly parallel efficiency that is greater than 95 % for up to 1024 PEs on a Hitachi SR2201 in problems with sufficiently large size data for each PE.

Iteration number for convergence of the iterative solver with localized preconditioning is increasing according to PE number. The localized preconditioning method was shown to be stabilized ASDD (additive Schwarz domain decomposition).

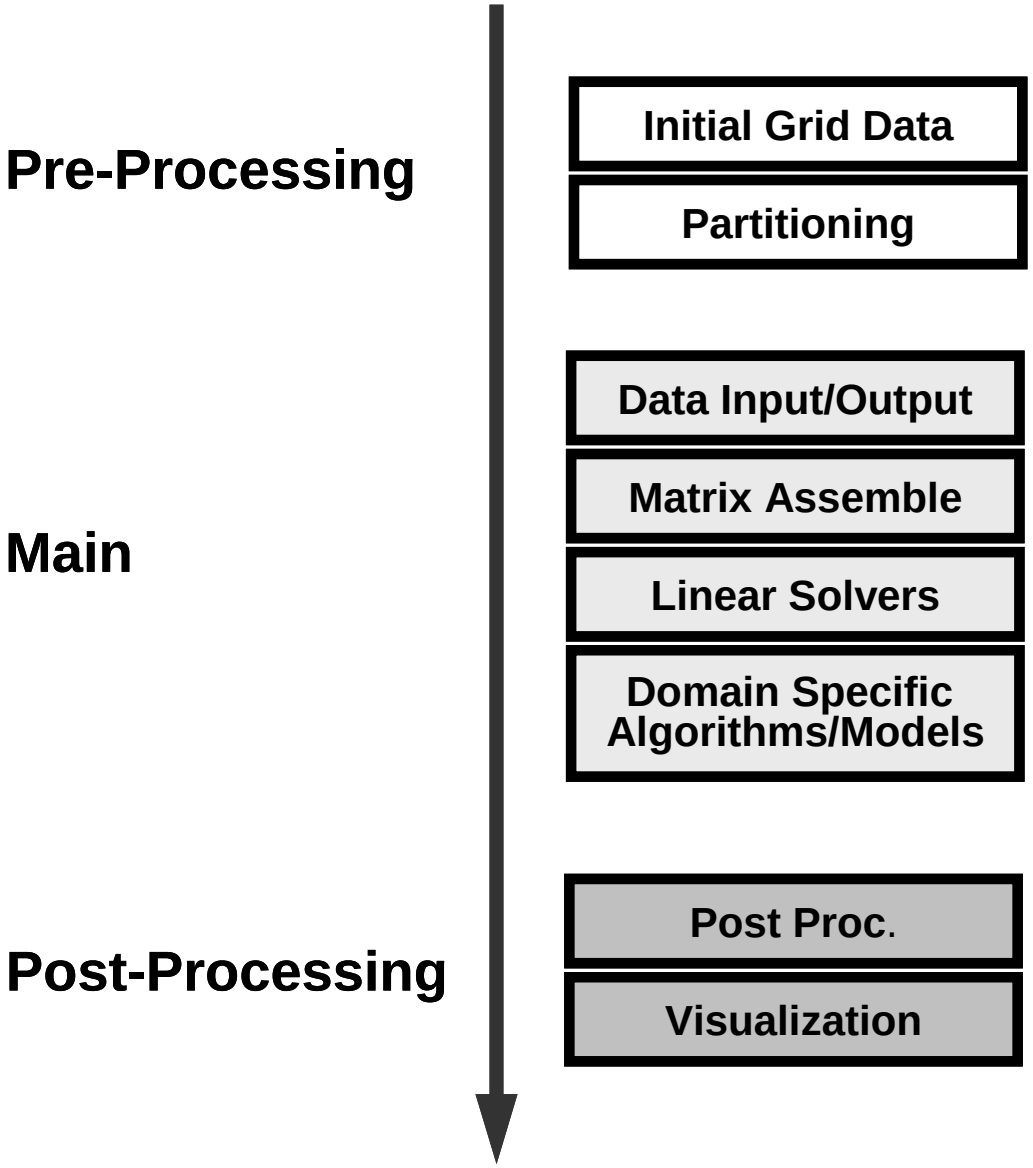


Fig. 2.1 Parallel FEM Procedure

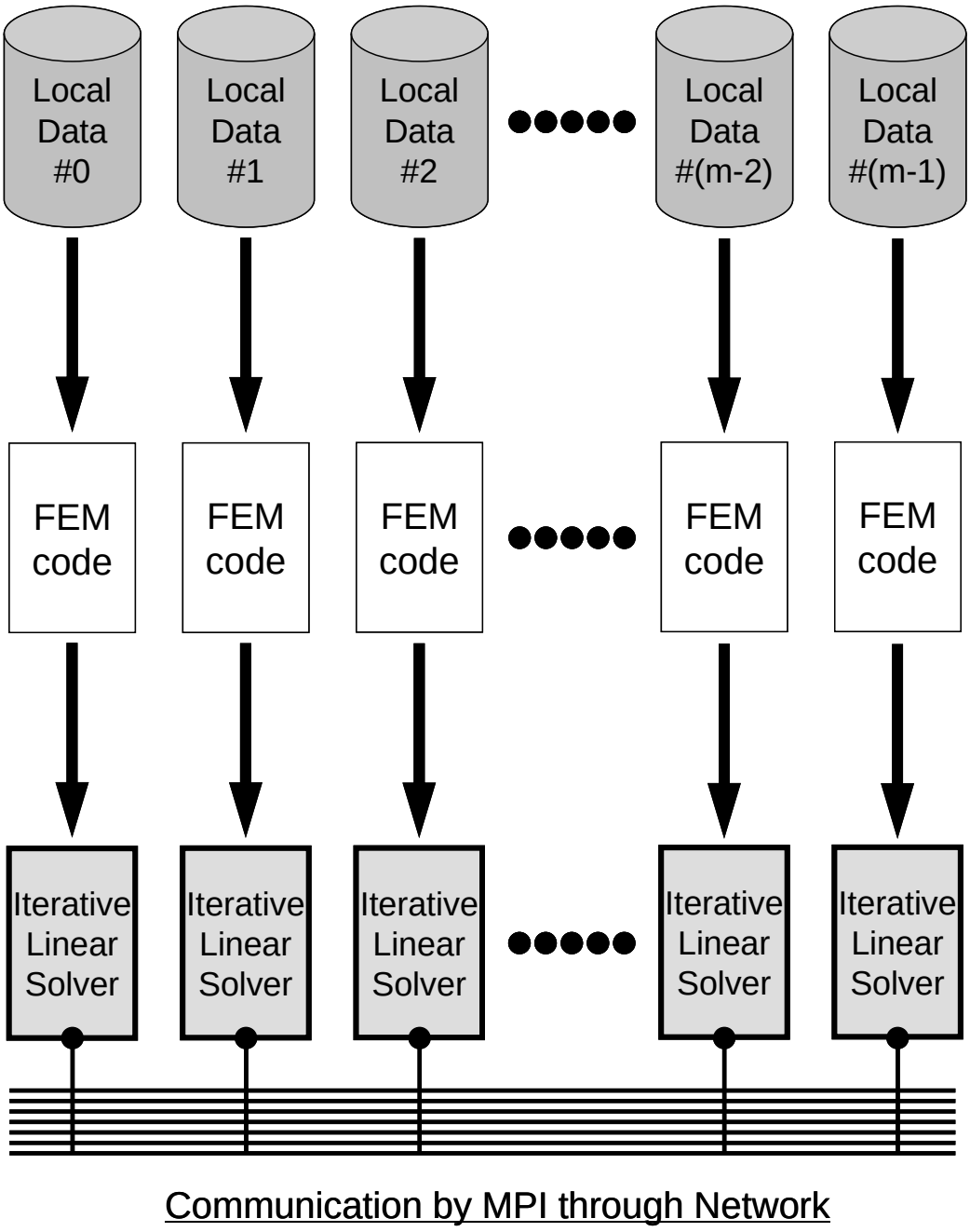


Fig. 2.2 Parallel FEM procedure and distributed local data sets in GeoFEM [71,72,73]

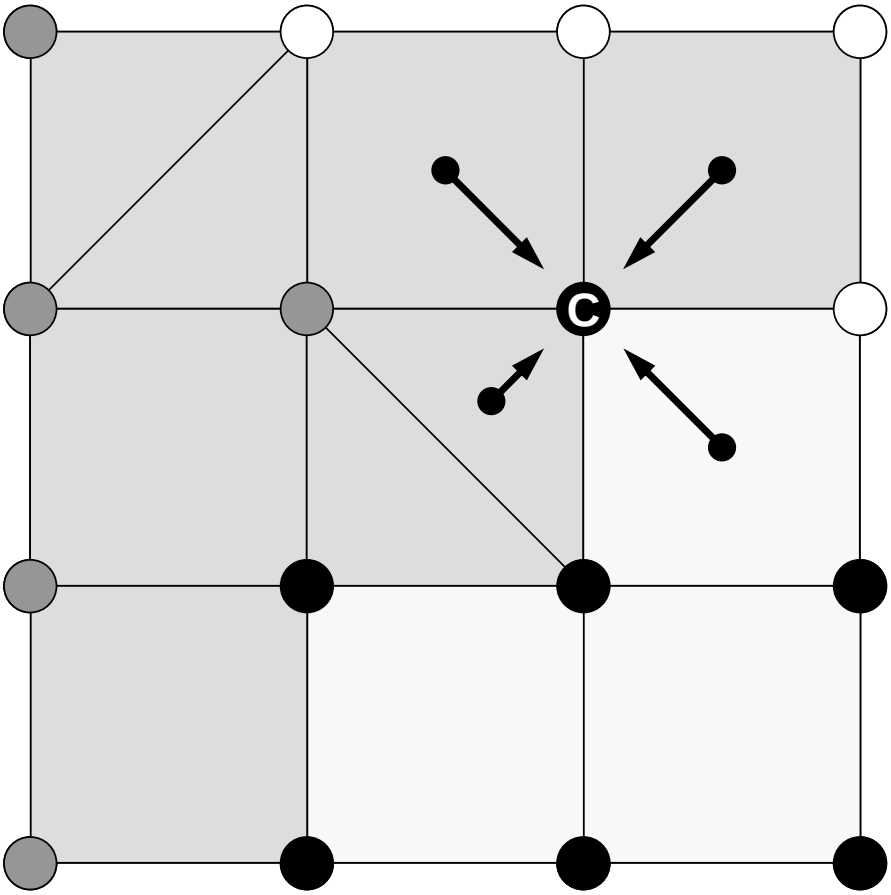


Fig. 2.3 *Element-by-element* operations around node *C*. Gray meshes are *overlapped* among domains.

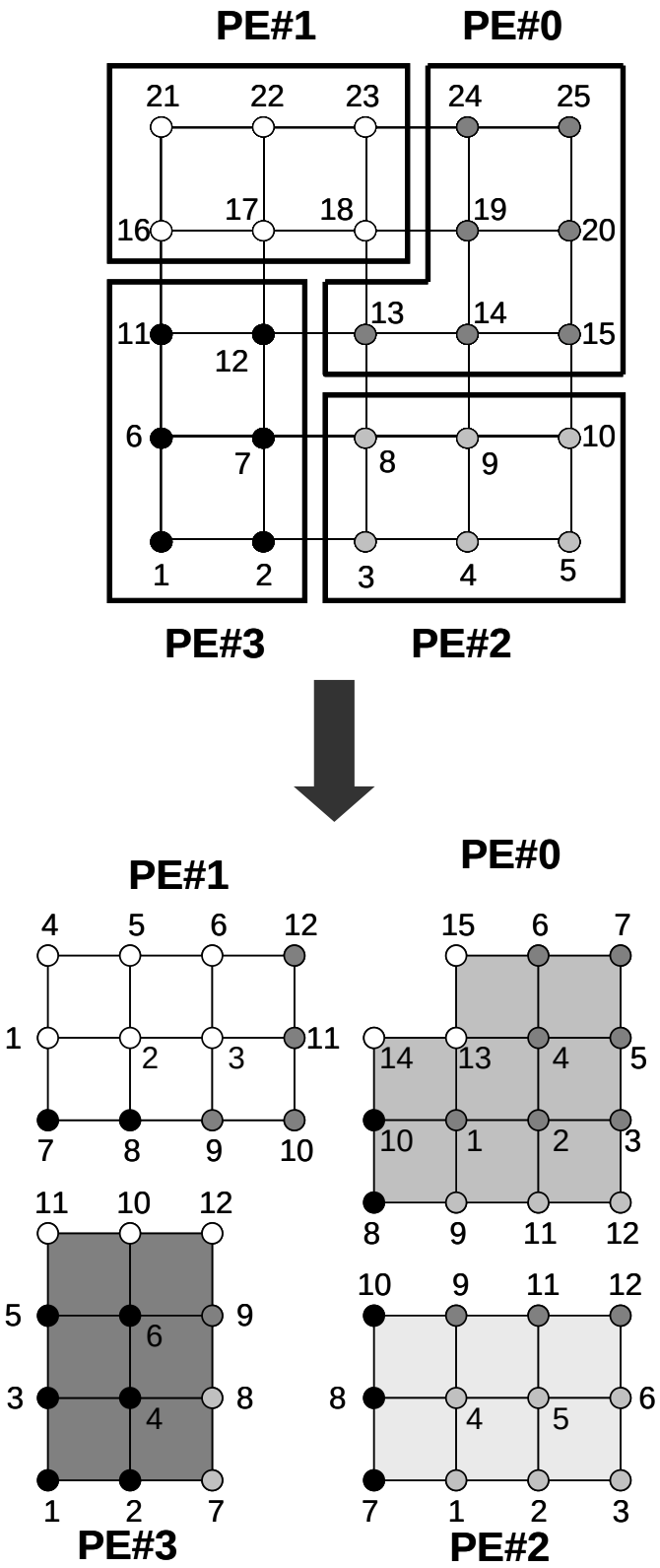


Fig. 2.4 Node-based partitioning into 4 PEs [71,72,73]

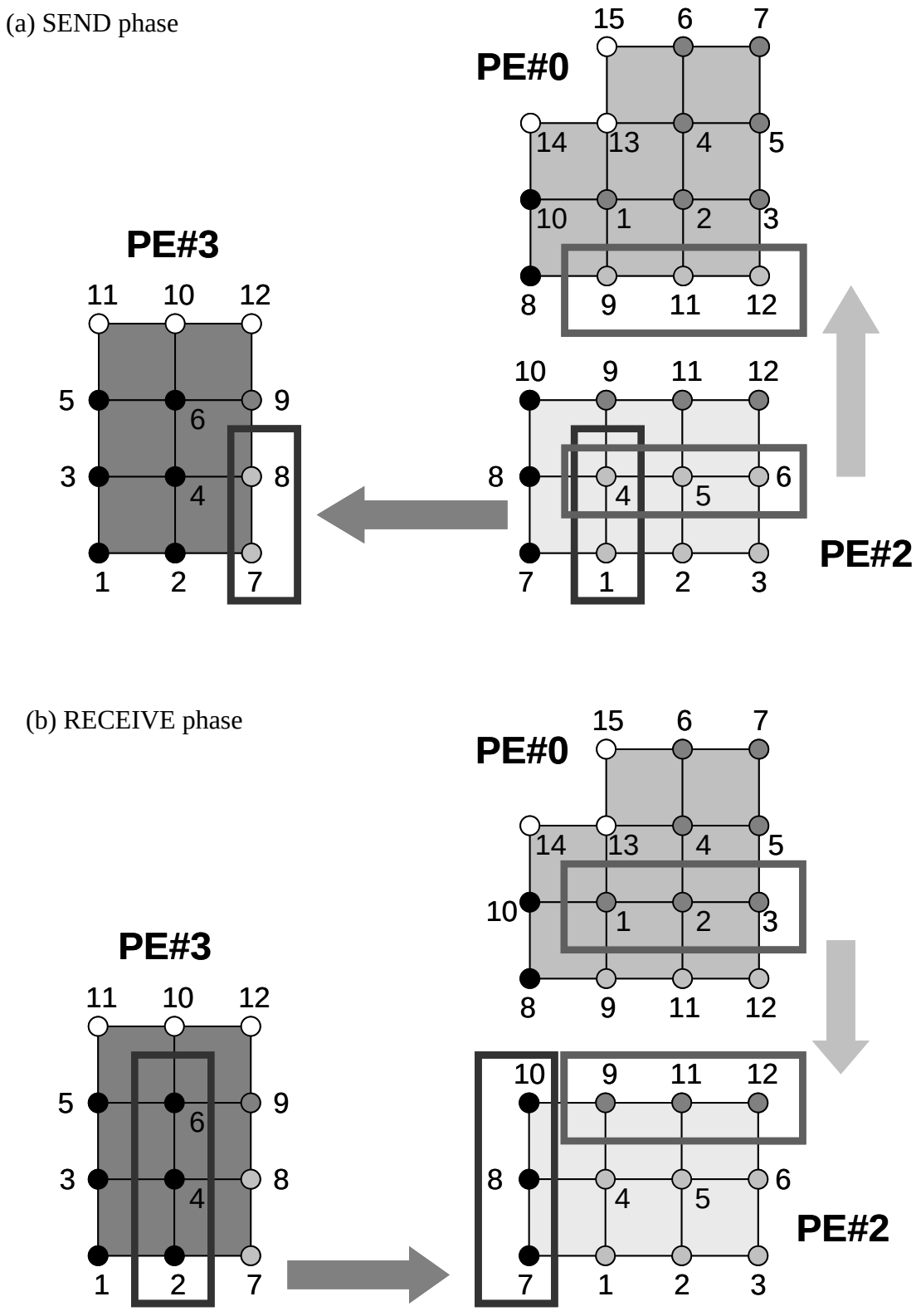


Fig. 2.5 Communication among processors [71,72,73]

(a) Calling interface for communication among domains (1x1 scalar and 3x3 block)

1x1 Scalar

```
      allocate (WS(NP), WR(NP), X(NP)
      call SOLVER_SEND_RECV                                          &
     &   ( NP, NEIBPETOT, NEIBPE, IMPORT_INDEX, IMPORT_NODE,         &
     &     EXPORT_INDEX, EXPORT_NODE, WS, WR, X , SOLVER_COMM,       &
     &     my_rank)
```

3x3 Block

```
      allocate (WS(3*NP), WR(3*NP), X(3*NP)
      call SOLVER_SEND_RECV_3                                        &
     &   ( NP, NEIBPETOT, NEIBPE, IMPORT_INDEX, IMPORT_NODE,         &
     &     EXPORT_INDEX, EXPORT_NODE, WS, WR, X , SOLVER_COMM,       &
     &     my_rank)
```

(b) Subroutines for communication among domains

- SEND phase

```
do neib= 1, NEIBPETOT
  istart= EXPORT_INDEX(neib-1)
  inum  = EXPORT_INDEX(neib  ) - istart
  do k= istart+1, istart+inum
    WS(k)= X(EXPORT_NODE(k))
  enddo
  call MPI_ISEND
       (WS(istart+1), inum, MPI_DOUBLE_PRECISION, &
        NEIBPE(neib), 0, SOLVER_COMM,             &
        req1(neib), ierr)
enddo
```

- RECEIVE phase

```
do neib= 1, NEIBPETOT
  istart= IMPORT_INDEX(neib-1)
  inum  = IMPORT_INDEX(neib  ) - istart
  call MPI_IRECV
       (WR(istart+1), inum, MPI_DOUBLE_PRECISION, &
        NEIBPE(neib), 0, SOLVER_COMM,             &
        req2(neib), ierr)
enddo

call MPI_WAITALL (NEIBPETOT, req2, sta2, ierr)

do neib= 1, NEIBPETOT
  istart= IMPORT_INDEX(neib-1)
  inum  = IMPORT_INDEX(neib  ) - istart
  do k= istart+1, istart+inum
    X(IMPORT_NODE(k))= WR(k)
  enddo
enddo

call MPI_WAITALL (NEIBPETOT, req1, sta1, ierr)
```

Fig. 2.6 Communication procedures among domains in GeoFEM [71,72,73,79,81]

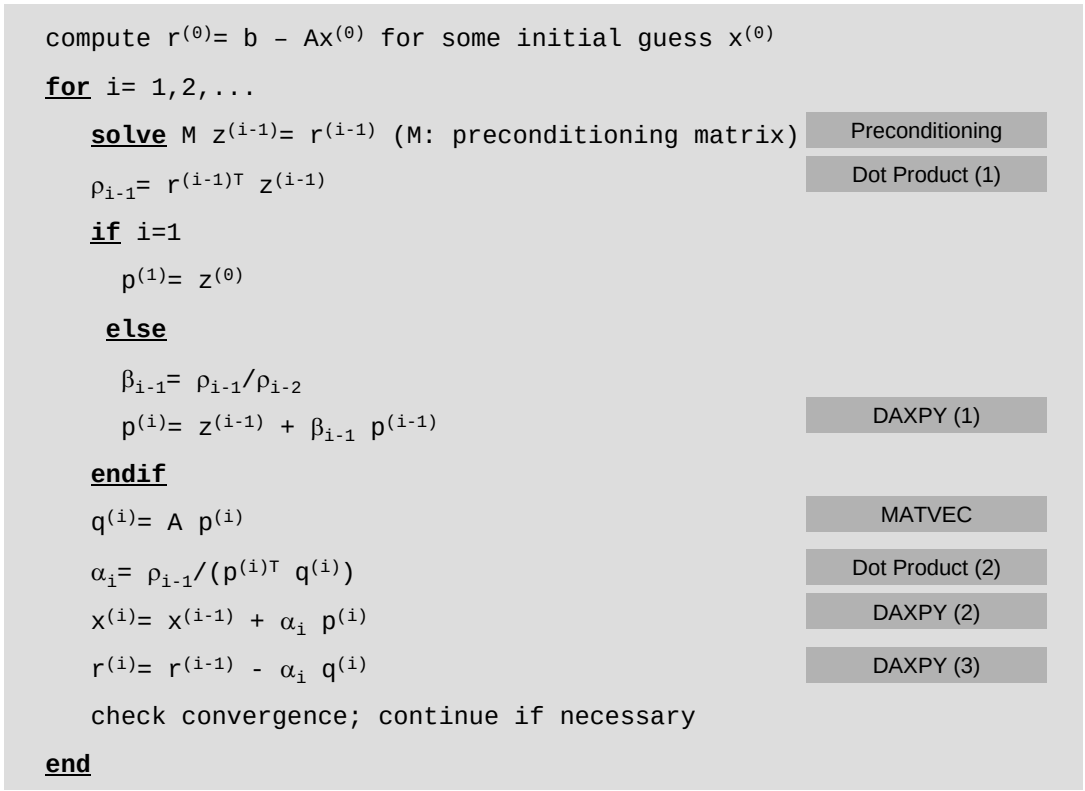


Fig. 2.7 Procedures in CG iterative method [7,21]

(a) Matrix-vector products

```
 do i= 1, N
   isL= INL(i-1) + 1
   ieL= INL(i  )
   WVAL= WW(i,R)
   do j= isL, ieL
     inod = IAL(j)
     WVAL=  WVAL -  AL(j) * WW(inod,Z)
   enddo
   WW(i,Z)= WVAL * DD(i)
 enddo

 do i= N, 1, -1
   SW = 0.0d0
   isU= INU(i-1) + 1
   ieU= INU(i  )
   do j= isU, ieU
     inod = IAU(j)
     SW= SW + AU(j) * WW(inod,Z)
   enddo
   WW(i,Z)= WW(i,Z) - DD(i) * SW
 enddo
```

(b) Inner dot products

```
 RHO0= 0.0
 do i= 1, N
   RHO0= RHO0 + WW(i,R)*WW(i,Z)
 enddo

 call MPI_allREDUCE (RHO0, RHO, 1, MPI_DOUBLE_PRECISION, &
&                    MPI_SUM, SOLVER_COMM, ierr)
```

(c) DAXPY

```
 do i= 1, N
   X (i)  = X (i)   + ALPHA * WW(i,P)
   WW(i,R)= WW(i,R) - ALPHA * WW(i,Q)
 enddo
```

Fig. 2.8 Parallelization of typical processes in iterative solvers in FORTRAN with MPI [71,72,73,79,81]

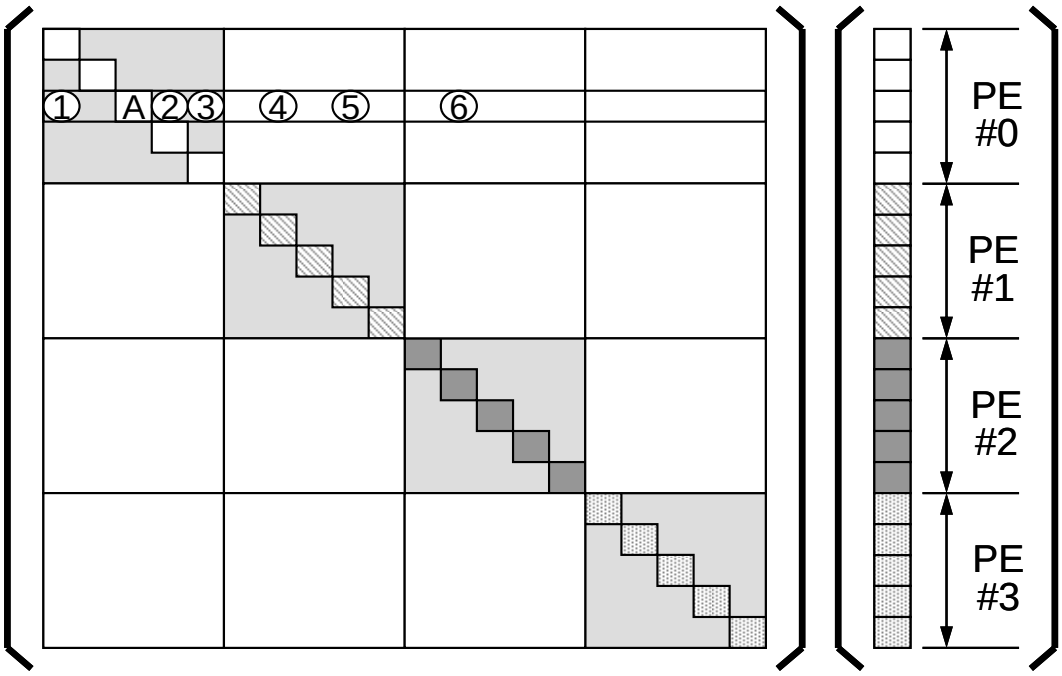


Fig.2.9 *Localized* ILU(0) Operation: Matrix components whose column numbers are outside the processor are ignored (set equal to 0) at *localized* ILU(0) factorization. For example the element *A* on PE#0 has 6 non-zero components but only 1,2,3 are considered and 4,5,6 are ignored and set to 0 [71,72,73,79,81]

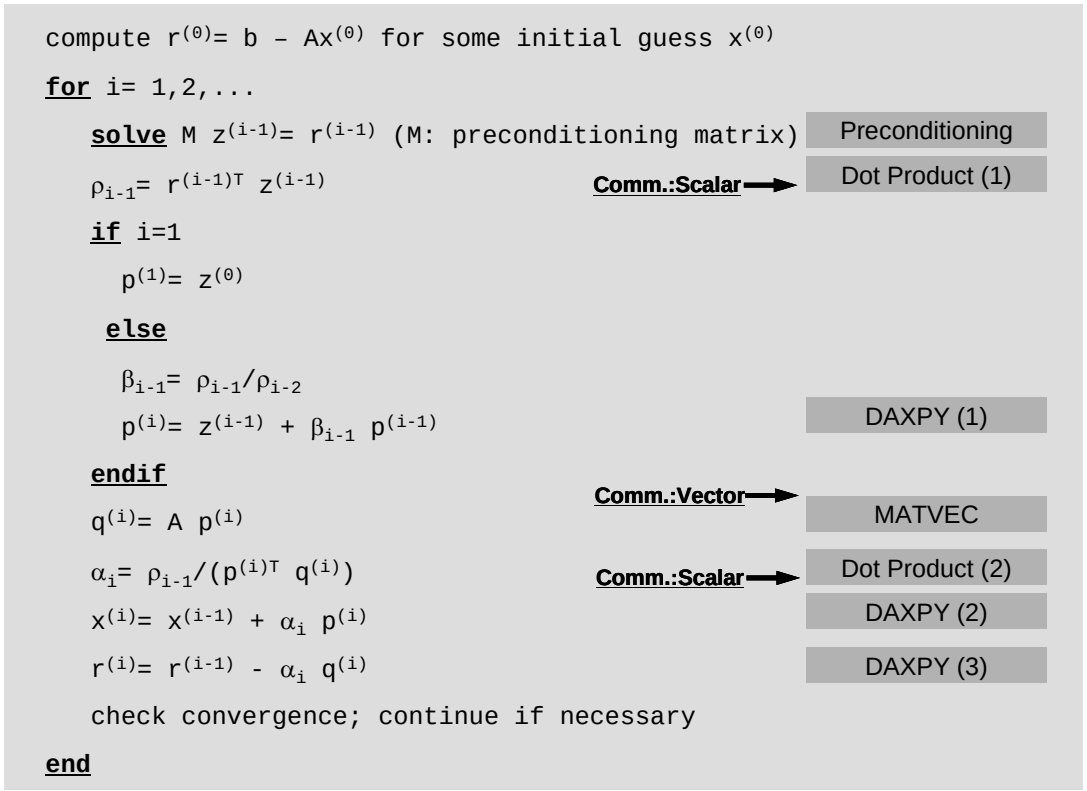


Fig. 2.10 Parallel CG iterative method by localized preconditioning in GeoFEM

Table 2.1 Homogeneous solid mechanics example with 3×44^3 DOF on Hitachi SR2201 solved by CG method with localized IC(0) preconditioning (Convergence Criteria $\varepsilon=10^{-8}$).

PE #	Iter. #	sec.	Speed Up
1	204	233.7	-
2	253	143.6	1.63
4	259	74.3	3.15
8	264	36.8	6.36
16	262	17.4	13.52
32	268	9.6	24.24
64	274	6.6	35.68

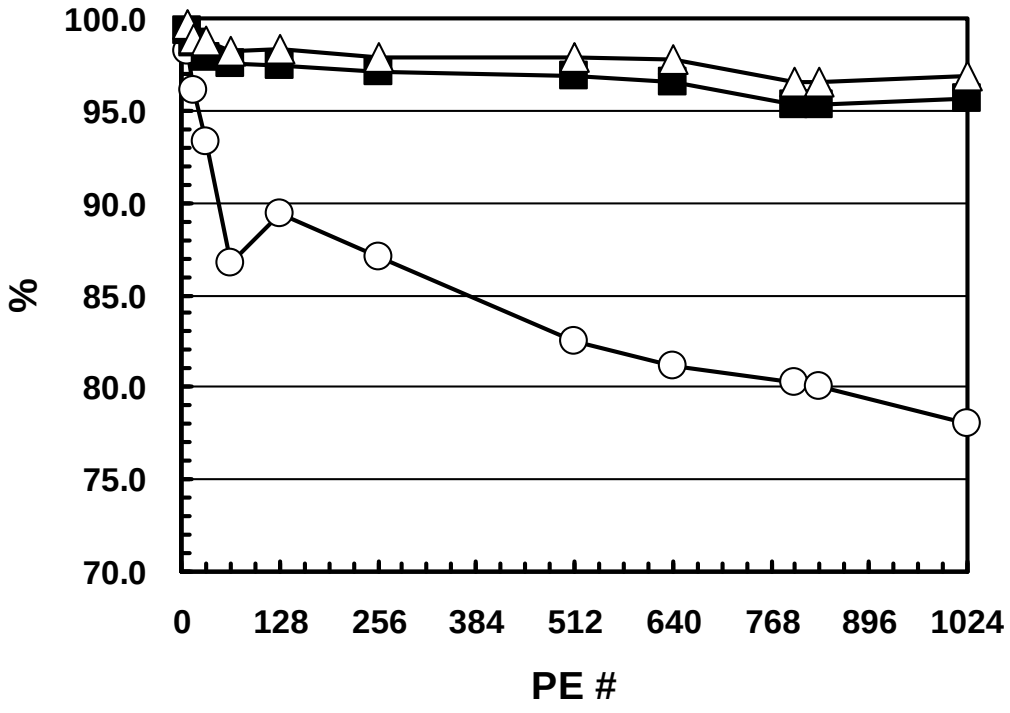
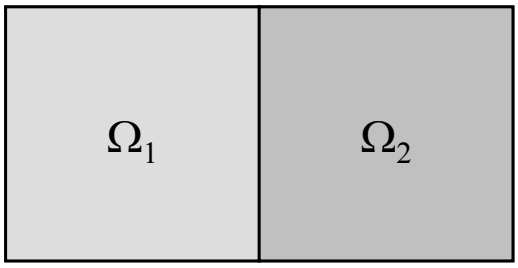


Fig. 2.11 Parallel performance for various problem sizes for simple 3D elastic solid mechanics on Hitachi SR2201. Problem size/PE is fixed. Largest case is 196,608,000 DOF on 1024 PEs. (Circles: 3×16^3 (= 12,288) DOF/PE, Squares: 3×32^3 (= 98,304), Triangles: 3×40^3 (= 192,000)).

(a) Local operation



(b) Global nesting correction

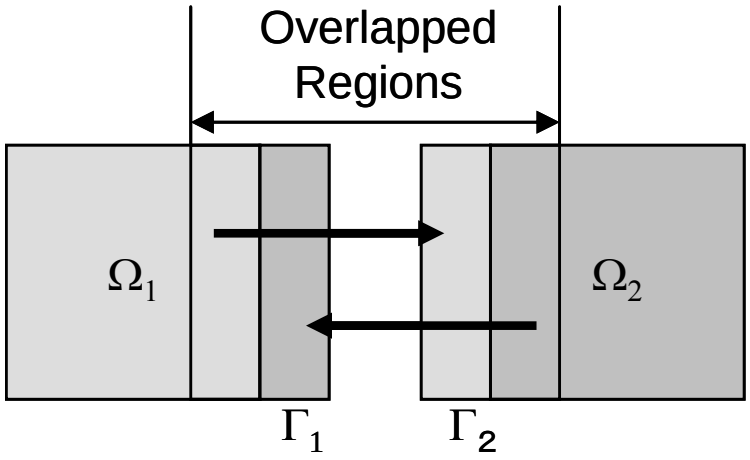


Fig.2.12 Operations in ASDD for 2 domains [107]

Table 2.2 Effect of ASDD for solid mechanics with 3×44^3 DOF on a Hitachi SR2201.

	NO Additive Schwarz			WITH Additive Schwarz		
PE #	Iter. #	Sec.	Speedup	Iter.#	Sec.	Speedup
1	204	233.7	-	144	325.6	-
2	253	143.6	1.63	144	163.1	1.99
4	259	74.3	3.15	145	82.4	3.95
8	264	36.8	6.36	146	39.7	8.21
16	262	17.4	13.52	144	18.7	17.33
32	268	9.6	24.24	147	10.2	31.80
64	274	6.6	35.68	150	6.5	50.07

Number of ASDD cycle/iteration = 1, Convergence Criteria $\varepsilon=10^{-8}$

Chapter 3 Parallel Iterative Solvers for Unstructured Grids using Hybrid Programming Model on SMP Cluster Architectures

An efficient parallel iterative method for unstructured grids has been developed on the GeoFEM platform for symmetric multiprocessor (SMP) cluster architectures with vector processors such as the Earth Simulator [130]. The method is based on a 3-level hybrid parallel programming model, including message passing for inter-SMP node communication, loop directives by OpenMP for intra-SMP node parallelization and vectorization for each processing element. Simple 3D linear elastic problems with more than 2.2×10^9 DOF have been solved using 3×3 block ICCG(0) method with additive Schwarz domain decomposition and PDJDS/CM-RCM (parallel DJDS/Cyclic Multicolor-Reverse Cuthil McKee) reordering on 176 nodes of the Earth Simulator, achieving performance of 3.80 TFLOPS. The PDJDS/CM-RCM reordering method provides excellent vector and parallel performance in SMP nodes. A three-level hybrid parallel programming model outperforms flat MPI in the problems involving large numbers of SMP nodes.

3.1 Introduction

In this chapter, parallel iterative methods have been developed for unstructured grids using a three-level *hybrid* parallel programming model for the Earth Simulator [130] on the GeoFEM platform. Individual PE of the Earth Simulator is a vector processor, therefore third-level of parallelism for vector processing should be considered in addition to the two levels in the *hybrid* parallel programming model, described in Chapter 1 (Fig.3.1). Following three levels of parallelism are considered:

- Inter-SMP node MPI for communication
- Intra-SMP node OpenMP for parallelization
- Individual PE compiler directives for vectorization

In *flat MPI* approach, communication among PEs through MPI and vectorization for individual PE have been considered for the Earth Simulator.

In the hybrid parallel programming model, the entire domain is partitioned into distributed local data sets [28,72,73,81], and each partition is assigned to one SMP node (Fig.3.2). On the contrast, each partition corresponds to each PE in the flat MPI.

In order to achieve efficient parallel/vector computation for applications with unstructured grids, the following 3 issues are critical:

- Local operation and no global dependency
- Continuous memory access
- Sufficiently long loops

A special reordering technique proposed by Washio et. al. [118,119] has been integrated with parallel iterative solvers with localized preconditioning developed in the GeoFEM project [118,119] in order to attain local operation, no global dependency, continuous memory access and sufficiently long loops.

In the following part of this chapter, we give an overview of special reordering techniques for parallel and vector computation on SMP nodes, and present the results for an application to 3D solid mechanics on the Earth Simulator [130]. Developed *hybrid* parallel programming model has been compared with the *flat MPI* programming model in Fig.3.1. Some of the results are compared with those obtained by Hitachi SR8000/128 at the University of Tokyo [133]. Hitachi SR8000 is also an SMP cluster

architecture. Each node consists of 8 PEs and provides 8 GFLOPS peak performance.

3.2 Reordering Methods for Parallel/Vector Performance on SMP Nodes

As shown in Fig.3.2, the entire domain is partitioned into local data sets for hybrid parallel programming model and each local data set is assigned to one SMP node.

3.2.1 Cyclic Multicolor – Reverse Cuthil McKee Reordering

In order to achieve efficient parallel/vector computation for applications with unstructured grids, the following 3 issues are critical:

- Local operations and no global dependency
- Continuous memory access
- Sufficiently long loops

For unstructured grids, in which data and memory access patterns are very irregular, reordering technique is very effective for achieving highly parallel and vector performance. The popular reordering methods are hyperplane/reverse Cuthil-McKee reordering and multicolor reordering [21,103]. The reverse Cuthil-McKee (RCM) method is a typical *level-set* ordering method. In Cuthill-McKee reordering, the elements of a level set are traversed from the nodes of lowest degree to those of highest degree according to dependency relationships, where the *degree* refers to the number of nodes connected to each node. In RCM, permutation arrays obtained in Cuthill-McKee reordering are reversed. RCM results in much less fill-in for Gaussian elimination and is suitable for iterative methods with IC or ILU preconditioning.

Multicolor reordering (MC) is much simpler than RCM. MC is based on an idea where no two adjacent nodes have the same color.

In both methods, elements located on the same hyperplane (or classified in the same color) are independent. Therefore, parallel operation is possible for the elements in the same hyperplane/color and the number of elements in the same hyperplane/color should be as large as possible in order to obtain high granularity for parallel computation or sufficiently large loop length for vectorization.

Hyperplane/RCM (Fig.3.3(a)) reordering provides fast convergence of IC/ILU-preconditioned Krylov iterative solvers, yet with irregular hyperplane size. For example in Fig.3.3(a), the 1st hyperplane is of size 1, while the 8th hyperplane is of size 8. In contrast, multicoloring provides a uniform element number in each color (Fig.

3.3(b)). However, it is widely known that the convergence of IC/ILU-preconditioned Krylov iterative solvers with MC reordering is rather slow. Convergence can be improved by increasing the number of colors, but this reduces the number of elements in each color.

The solution for this trade-off is cyclic multicolor reordering (CM) on hyperplane/RCM [118,119]. In this method, the hyperplanes are renumbered in a cyclic manner. Figure 3.3 (c) shows an example of CM-RCM reordering. In this case, there are 4 colors ; the 1st, 5th, 9th and 13th hyperplanes in Fig.3.3 (a) are classified into the 1st color. There are 16 elements in each color. In CM-RCM, the number of colors should be large enough to ensure that elements in the same color are independent.

In this study, implementation of MC is also considered and is compared with CM-RCM.

3.2.2 DJDS Reordering

The compressed row storage (CRS) [7,103] matrix storage format originally used in GeoFEM is highly memory-efficient, however the innermost loop is relatively short due to matrix-vector operations as follows (Fig.3.4):

```
do i= 1, N
  do j= 1, NU(i)
     k1= indexID(i,j); k2= itemID(k1)
     F(i)= F(i) + A(k1)*X(k2)
  enddo
enddo
```

The following loop exchange is then effective for obtaining a sufficiently long innermost loop (Fig.3.4):

```
do j= 1, NUmax
  do i= 1, N
     k1= indexID(i,j); k2= itemID(k1)
     F(i)= F(i) + A(k1)*X(k2)
  enddo
enddo
```

Descending-order jagged diagonal storage (DJDS) [81,118,119] is suitable for this type of operation and involves permuting rows into an order of decreasing number of non-zeros, as shown in Fig.3.5 (a). As elements on the same hyperplane are independent, performing this permutation inside a hyperplane does not affect results. Thus, a 1D array of matrix coefficients with continuous memory access can be obtained, as shown in Fig.3.5 (b).

3.2.3 Distribution over SMP Nodes : Parallel DJDS Reordering

The 1D array of matrix coefficients with continuous memory access is suitable for both parallel and vector computing. The loops for this type of array are easily distributed to each PE in an SMP node via loop directives. In order to balance the computational load across PEs in the SMP node, the DJDS array should be reordered again in cyclic manner. The procedure for this reordering, called parallel DJDS (PDJDS) is described in Fig.3.6.

3.2.4 Summary of Reordering Methods

The reordering procedures for increasing parallel/vector performance of the SMP cluster architecture described in this section are summarized as follows:

（1） RCM reordering on the original local matrix for independent sets.
（2） CM reordering to obtain loops whose length is sufficiently long and uniform.
（3） DJDS reordering for efficient vector processing, producing 1D arrays of coefficients with continuous memory access and long loops.
（4） Cyclic reordering for load-balancing among PEs on an SMP node.
（5） PDJDS/CM-RCM reordering is complete.

Figure 3.7 and 3.8 show the procedure for forward/backward substitution procedure during ILU(0)/IC(0) preconditioning by PDJDS/CM-RCM reordering. In the *flat MPI* programming model, `PEsmpTOT` in Fig.3.7 is set to 1 without any option of OpenMP for compiler while `PEsmpTOT` is set to 8 in the *hybrid* programming model. Figure 3.9 shows the typical procedures of the iterative methods, such as matrix-vector products, inner dot products and DAXPY, based on the OpenMP and vectorization directives on the Earth Simulator.

3.3 Vector and Parallel Performance in Simple Geometries

The proposed methods were applied to 3D solid mechanics example cases, as described in Fig.3.10, which represent linear elastic problems with homogeneous material properties and boundary conditions. Each element is a tri-linear (1st-order) cubic hexahedral element with unit edge length, and each node has 3 DOF, therefore there are 3×Nx×Ny×Nz DOF in total for the problem (Fig.3.10).

For this problem, 3×3 Block ICCG(0) with PDJDS/CM-RCM reordering is applied with full LU factorization for each 3×3 diagonal block. One ASDD operation is applied to each iteration. In each case, the number of colors for CM reordering was set to 99, corresponding to an average vector length of (total number of FEM nodes)/(99 × NPE), where NPE is the number of PEs on each SMP node (1 for flat MPI, and 8 for hybrid programming model).

The increase in speed for a fixed problem size (3×128^3=6,291,456 DOF) using between 1 and 8 SMP nodes was evaluated for the hybrid and flat MPI programming models. Figure 3.11 shows the results. The number of iterations for convergence ($\varepsilon=10^{-8}$) was 333 (1-node), 337 (2-nodes), 338 (4-nodes), and 341 (8-nodes) for the hybrid programming model, and 341(1-node), 344(2-nodes), 348(4-nodes), and 352(8-nodes) for the flat MPI indicating that the number of iterations remains almost constant as the number of nodes increases. This is due to the ASDD. The speedup rate for 8 SMP nodes was 5.78 (hybrid) and 6.36 (flat MPI), which corresponds to 72.2% and 79.5% of the linear (ideal) speedup. The speedup effect for many nodes is worse than the ideal speedup due to the smaller problem size per node. The performance for 1 node (8 PEs) were 21.9 GFLOPS (34.2% of peak performance of 64 GFLOPS) for the hybrid model and 23.4 GFLOPS (36.6% of peak performance) for flat MPI. While both the hybrid and flat MPI models provide good vector and parallel performance, the flat MPI gives slightly better results.

3.4 Effect of Reordering

3.4.1 Vector Performance

The effect of PDJDS/CM-RCM reordering for the vector performance on the Earth Simulator was evaluated. The performance of PDJDS/CM-RCM is compared with the original block ICCG solver in GeoFEM [81,118,119], in which components of coefficient matrices are stored in the CRS manner without reordering. Figure 3.12 shows the performance for a fixed problem size (3×64^3=786,432 DOF) using between 1 and 8 PEs was evaluated for the flat MPI programming models. The performance of the ICCG solver is dramatically improved by PDJDS/CM-RCM reordering. Single PE performance of the original ICCG solver is only 0.17 GFLOPS, corresponding to 2.13% of the peak performance. PDJDS/CM-RCM reordering provides 3.22 GFLOPS (40.3% of peak performance). Speed-up ratio for 8 PEs is 7.63 by original solver and 6.40 by PDJDS/CM-RCM reordering. This is because the ratio of communication overhead to the entire process increases due to the improvement of computation process by PDJDS/CM-RCM reordering.

3.4.2 SMP Parallel Performance by Hybrid Parallel Programming Model

Figure 3.13 shows the results demonstrating the effect of PDJDS/CM-RCM reordering for the *hybrid* programming model. In this case, the following 3 cases were compared (Fig.3.14):

- PDJDS/CM-RCM reordering
- Parallel descending-order compressed row storage (PDCRS) /CM-RCM reordering
- CRS without reordering

PDCRS/CM-RCM reordering is identical to PDJDS/CM-RCM except that the matrices are stored in a CRS manner [7,103] after permutation of rows into the order of decreasing number of non-zeros. The length of the innermost loop is shorter than that for PDJDS. The elapsed execution time was measured for various problem sizes from 3×16^3 (12,288) DOF to 3×128^3 (6,291,456) DOF on one SMP node of the Earth Simulator (8 PEs, 64 GFLOPS peak performance, 16 GB memory) and Hitachi SR8000/128 (8 PEs, 8 GFLOPS peak performance, 8 GB memory). The difference between PDCRS and PDJDS for smaller problems is not significant, but PDJDS

outperforms PDCRS for larger problems due to longer vector length. On the Earth Simulator, the PDCRS performs at a steady 1.5 GLOPS (2.3% of peak performance), while the performance of PDJDS increases from 3.81 GFLOPS to 22.7 GFLOPS with problem size. Results on the Hitachi SR8000/128 show similar feature due to pseudo-vectorization of the compiler [133], but performance of PDCRS increases with problem size on Hitachi SR8000/128.

The cases without reordering exhibit very poor performance of only 0.30 GFLOPS (Earth Simulator, 0.47% of peak performance) and 0.09 GFLOPS (Hitachi SR8000, 1.13% of peak performance). Parallel computation is impossible for forward/backward substitution (FBS) in the IC factorization process even in the simple geometry examined in this study. This FBS process represents about 50% of the total computation. If this process is not parallelized, the performance decreases significantly.

Figure 3.15 shows communication/synchronization overhead inside SMP node for various problem sizes using one SMP node. Communication/synchronization overhead occurs for parallel processing in each SMP node. The work ratio was measured for various problem sizes from 3×16^3 (12,288) DOF to 3×128^3 (6,291,456) DOF on one SMP node on the Earth Simulator and Hitachi SR8000/128. Overhead for intra-SMP node communication becomes small if the problem size is larger. Figure 3.15 shows speed-up ratio from one PE to one SMP node (8 PEs). Results for large problem are 7.01 (Earth Simulator) and 7.40 (Hitachi SR8000/128), respectively.

3.4.3 Effect of Reordering Method

The block ICCG(0) solver with PDJDS/CM-RCM reordering exhibited excellent performance on the Earth Simulator for the simple geometry and boundary conditions, as shown in Fig.3.10. In complicated geometries for real-world applications, the number of hyperplanes may be extremely large [81], making it very difficult to construct independent sets with sufficient loop length by CM reordering. In this situation, classical multicolor reordering (MC), as described in Fig.3.3 (b), is a reasonable alternative. MC usually provides slower convergence than CM-RCM or RCM but sufficient loop length is guaranteed when the number of colors is specified. In the tests described in this section, the PDJDS/MC reordering method was applied to various types of computations, and the obtained results are compared to those for PDJDS/CM-RCM. In PDJDS/MC reordering, MC, rather than CM-RCM, is applied in the 1st and 2nd stage of the reordering procedures. All of the computations in these tests were performed on one SMP node of the Earth Simulator and a Hitachi SR8000/128 according to the *hybrid* parallel programming model.

In the first example, 3D linear elastic problems in Fig.3.10 with 3×128^3=6,291,456 DOF were solved. Figures 3.16-3.18 show the results obtained by Block ICCG(0) solver using these two reordering methods (PDJDS/CM-RCM and PDJDS/MC). PDJDS/CM-RCM generally exhibits better performance with regard to both CPU time and GFLOPS rate. In cases with many colors, fewer iterations are required for convergence, but the performance is worse due to the smaller loop length and greater overhead. As shown in Fig.3.16, the number of iterations in PDJDS/MC does not change with the number of colors if more than 500 are defined, whereas the CPU time increases (Fig.3.17). Although the characteristics of the results obtained using the Earth Simulator and those obtained using the Hitachi SR8000/128 appear similar, the Earth Simulator is more sensitive to decrease in loop length. The performance of the Earth Simulator is significantly degraded if the number of colors is more than 400, as shown in Fig.3.18. This is because loop length is less than the size of vector register, which is 256.

In the next example, the 3D linear elastic problem was solved for a complicated geometry of a micro pin-grid array (PGA) model for mobile computers [143]. Figure 3.19 shows the mesh configuration for the problems solved in this section, having 61 pins, 956,128 elements and 1,012,354 nodes (3,037,062 DOF) [22]. The mesh is generated according to the information in [143]. Figure 3.20 shows the stress intensity distribution visualized by parallel volume rendering method in GeoFEM [15]. Table 3.1 and Fig.3.21-3.22 compares PDJDS/CM-RCM and PDJDS/MC. In this complicated geometry, PDJDS/MC provides better performance in terms of CPU time and GFLOPS rate than does PDJDS/CM-RCM. In PDJDS/CM-RCM, the number of colors for independent sets is extremely large (more than 1,000 even after application of the rooting method by Gibbs [83]). Therefore, more CPU time is required for convergence due to the smaller loop length and higher overhead, even though the number of iterations for convergence is smaller than PDJDS/MC. The Earth Simulator is also more sensitive to decrease in loop length, as shown in Fig.3.22.

3.5 Performance Evaluation for Large Scale Problems

Figures 3.23-3.35 show the results for large-scale problems having simple geometries and boundary conditions as in Fig.3.10 implemented on up to 176 SMP nodes of the Earth Simulator (1,408 PEs, 11.26 TFLOPS peak performance, 2.8 TB memory). The hybrid and flat MPI models were evaluated. The problem size for one SMP node was fixed and the number of nodes was varied between 1 and 176. The largest problem size was 176×3×128×128×256 (2,214,592,512) DOF, for which the performance was about 3.80 TFLOPS, corresponding to 33.7 % of the total peak performance of the 176 SMP nodes. The parallel work ratio among SMP nodes for MPI is more than 90% if the problem is sufficiently large. The PDJDS/CM-RCM reordering has been applied to all cases.

The performance of the hybrid model is competitive with that of the flat MPI model, and both provide robust convergence and good parallel performance for a wide range of problem sizes and SMP node numbers. Iterations for convergence in the hybrid and flat MPI are almost equal, although the hybrid converges slightly faster, as shown in Fig.3.28 (a) and Fig.3.34 (a). In general, flat MPI performs better for the hybrid model for smaller numbers of SMP nodes, as shown in Fig.3.23-3.28, while the hybrid outperforms flat MPI when a large number of SMP nodes are involved (Fig.3.29-3.35), especially if the problem size per node is small, as shown in Fig.3.29, 3.30, 3.34 and 3.35. This is due to the increase in overhead for communications.

Figure 3.36 shows time spent for communication subroutines per iteration by the flat MPI programming model on the Earth Simulator using between 8 and 176 SMP nodes, where the problem size/SMP node is fixed as 6,291,456 DOF (3×128^3). The maximum elapsed time increases with SMP node number, although the data size of communication for each PE and the number of neighboring PEs remain constant in this type of homogeneous problem, as shown in Fig.3.10. This is mainly because of the latency for MPI communication. According to the performance estimation for finite-volume application code for CFD with local refinement in [45], a greater percentage of time is required by the latency component on larger processor counts, simply due to the available bandwidth being much larger (Fig.3.37). Flat MPI requires eight times as many MPI processes as hybrid model. If the node number is large and problem size is small, this effect is significant.

3.6 Summary

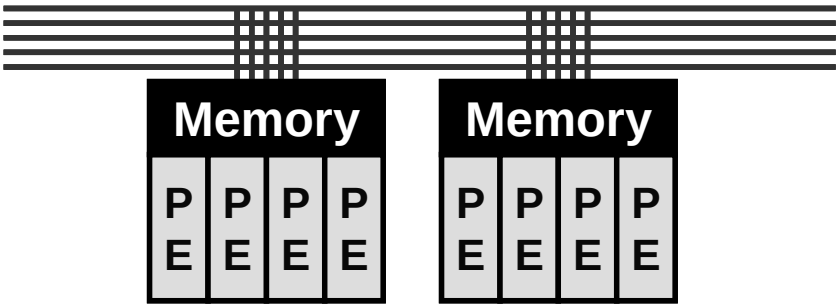
This chapter described an efficient parallel iterative method for unstructured grids developed for the GeoFEM platform on SMP cluster architectures with vector processors such as the Earth Simulator. The method employs a *three-level hybrid* parallel programming model consisting of the following hierarchy:

- Inter-SMP node MPI
- Intra-SMP node OpenMP for parallelization
- Individual PE Compiler directives for vectorization

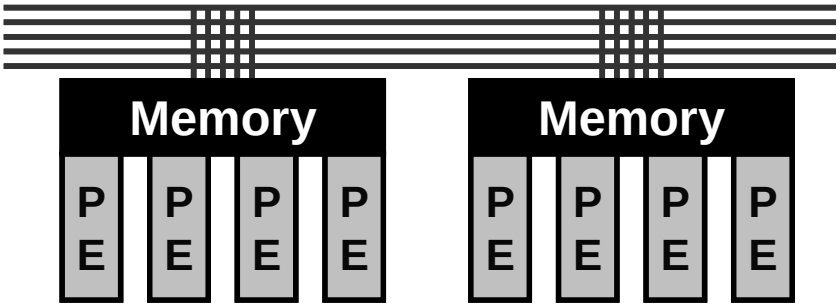
Simple 3D linear elastic problems with more than 2.2×10^9 DOF were solved by 3×3 block ICCG(0) with additive Schwarz domain decomposition and PDJDS/CM-RCM reordering on 176 SMP nodes of the Earth Simulator, achieving a performance of 3.80 TFLOPS (33.7 % of peak performance). PDJDS/CM-RCM reordering provides excellent vector and parallel performance on SMP nodes. Without reordering, parallel processing of forward/backward substitution in IC/ILU factorization was impossible due to global data dependencies even in the simple examples in this study. Although the three-level hybrid and flat MPI parallel programming models offer similar performance, the hybrid programming model outperforms flat MPI in problems with a large numbers of SMP nodes.

The performance of PDJDS/CM-RCM reordering was also compared with PDJDS/MC. In a simple cubic geometry, PDJDS/CM-RCM usually converges faster than PDJDS/MC. However, when complicated geometries are involved with a large number of hyperplanes, in which case it is difficult to construct independent sets with sufficient loop lengths by CM, PDJDS/MC provides better performance in terms of GFLOPS rate and CPU time by guaranteeing sufficient loop length, even though PDJDS/CM-RCM requires fewer iterations for convergence.

The most appropriate reordering method should therefore be selected based on the length of each hyperplane generated by RCM reordering.



Hybrid : Hierarchy



Flat-MPI : Each PE -> Independent

	Each PE	Intra NODE	Inter NODE
Hybrid	F90 + directives (OpenMP)		MPI
Flat-MPI	F90	MPI	

Fig. 3.1 Parallel programming models for SMP cluster architectures [13,20,23,81,86]

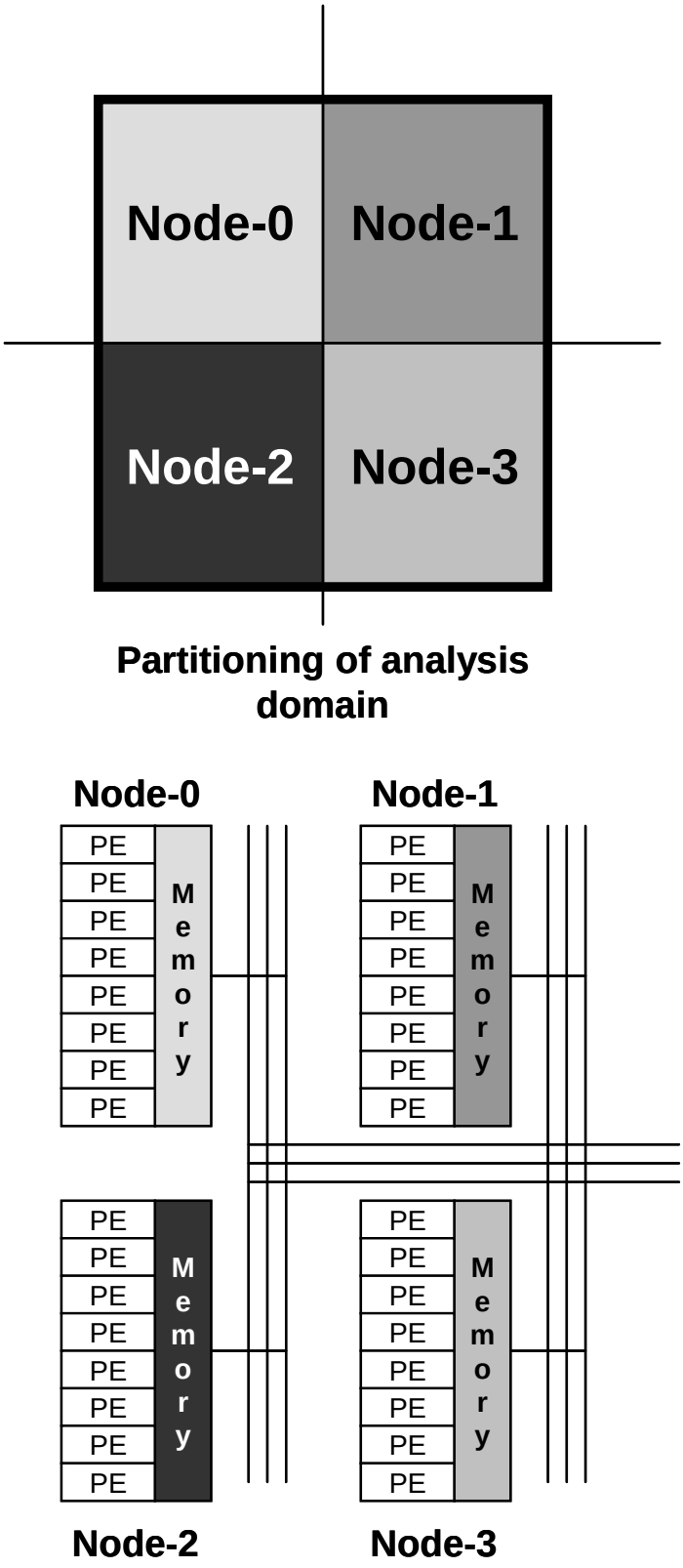


Fig. 3.2 Parallel FEM computation on SMP cluster architecture using *hybrid* parallel programming model. Each partition corresponds to an SMP node [81]

(a) Hyperplane/RCM

8	9	10	11	12	13	14	15
7	8	9	10	11	12	13	14
6	7	8	9	10	11	12	13
5	6	7	8	9	10	11	12
4	5	6	7	8	9	10	11
3	4	5	6	7	8	9	10
2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8

(b) Multicoloring: 4 colors

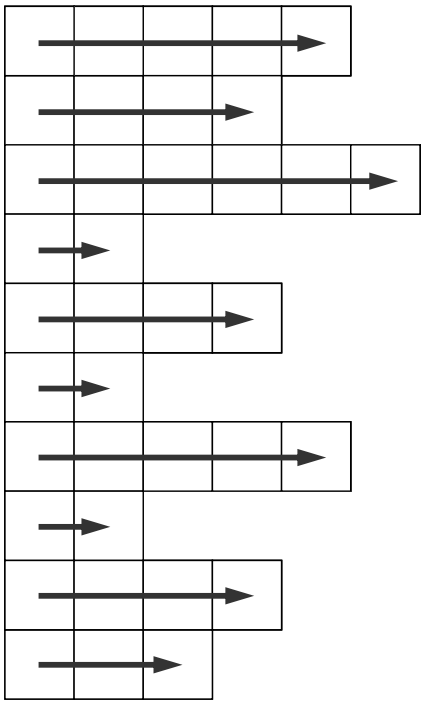
3	4	3	4	3	4	3	4
1	2	1	2	1	2	1	2
3	4	3	4	3	4	3	4
1	2	1	2	1	2	1	2
3	4	3	4	3	4	3	4
1	2	1	2	1	2	1	2
3	4	3	4	3	4	3	4
1	2	1	2	1	2	1	2

(c) CM-RCM: 4 colors

4	1	2	3	4	1	2	3
3	4	1	2	3	4	1	2
2	3	4	1	2	3	4	1
1	2	3	4	1	2	3	4
4	1	2	3	4	1	2	3
3	4	1	2	3	4	1	2
2	3	4	1	2	3	4	1
1	2	3	4	1	2	3	4

Fig. 3.3 Example of hyperplane/RCM, multicoloring and CM-RCM reordering for 2D geometry [81]

1D-Storage
memory saved, short vector length



2D-Storage w/Hyperplane/RCM
long vector length, many ZERO's

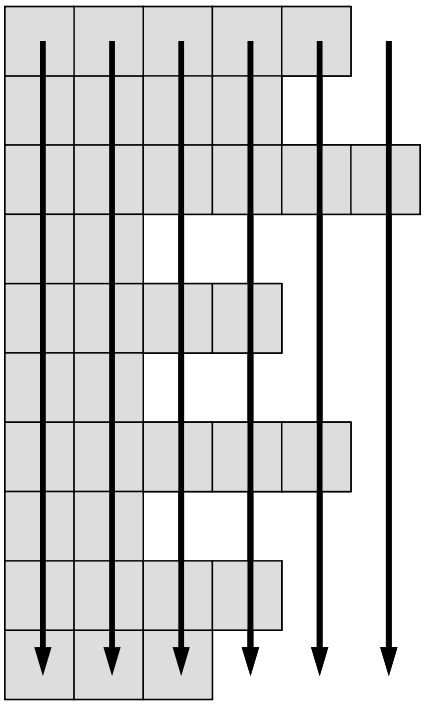
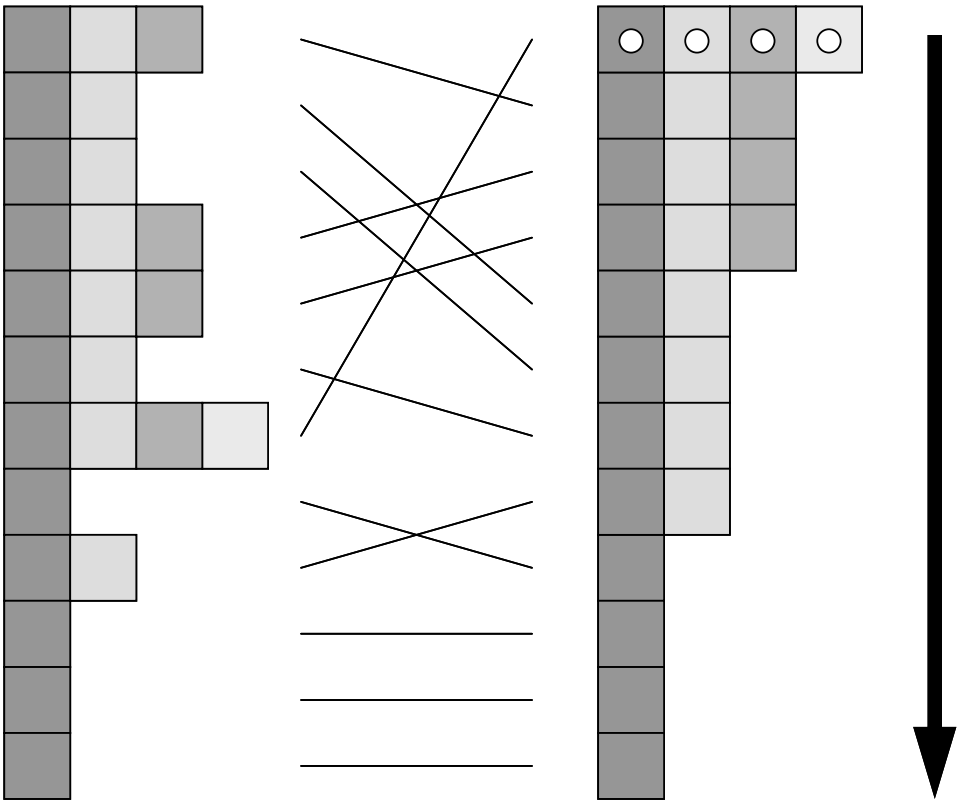
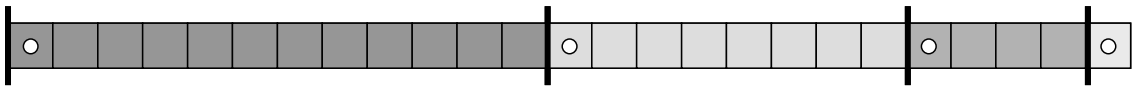


Fig. 3.4 1D and 2D storage of coefficient matrices for linear equations [81]

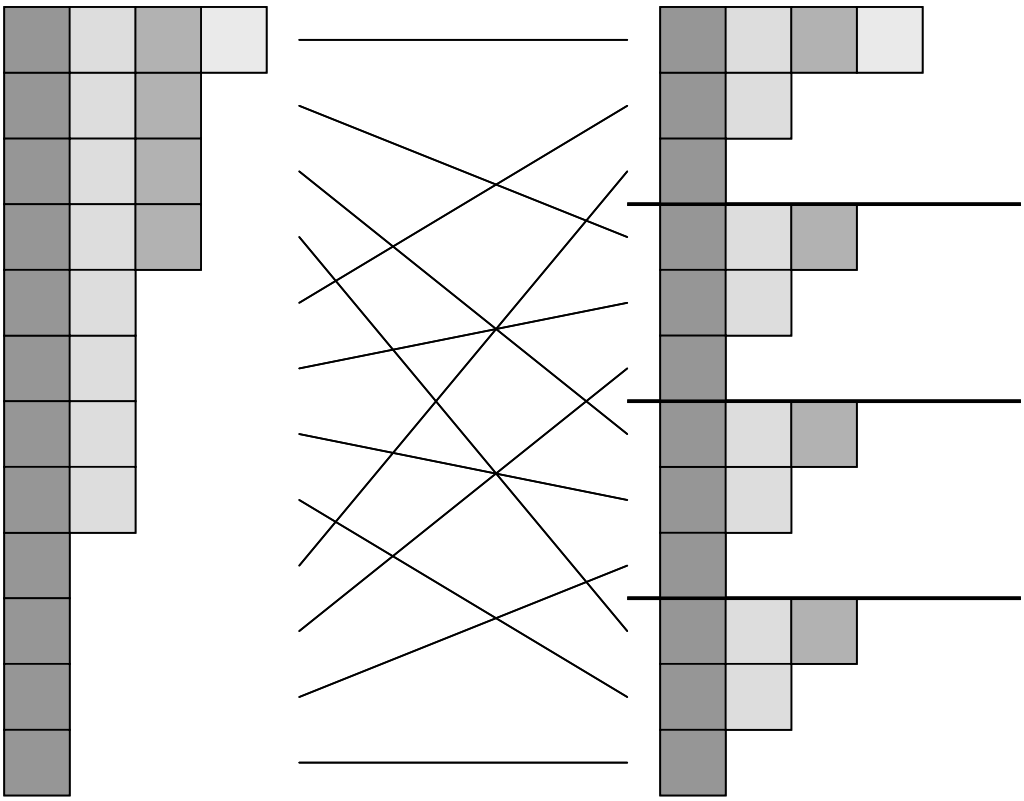


(a) Permutation of rows into order of decreasing number of non-zeros

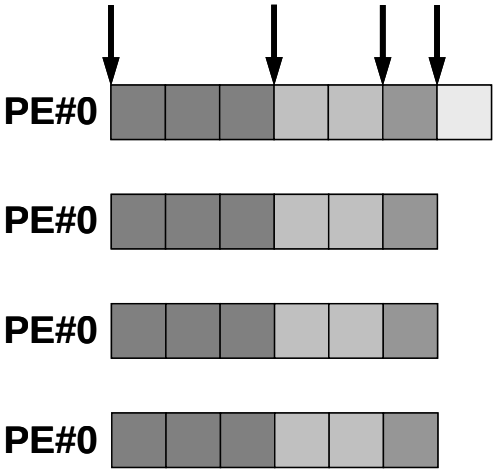


(b) 1D array of matrix coefficient

Fig. 3.5 DJDS reordering for efficient vector/parallel processing [81,118,119]



(a)



(b)

Fig. 3.6 PDJDS reordering for an SMP node : Example with 4 PEs per SMP node (a) Cyclic reordering (b) 1D array assigned to each PE after reordering and load-balancing [81,118,119]

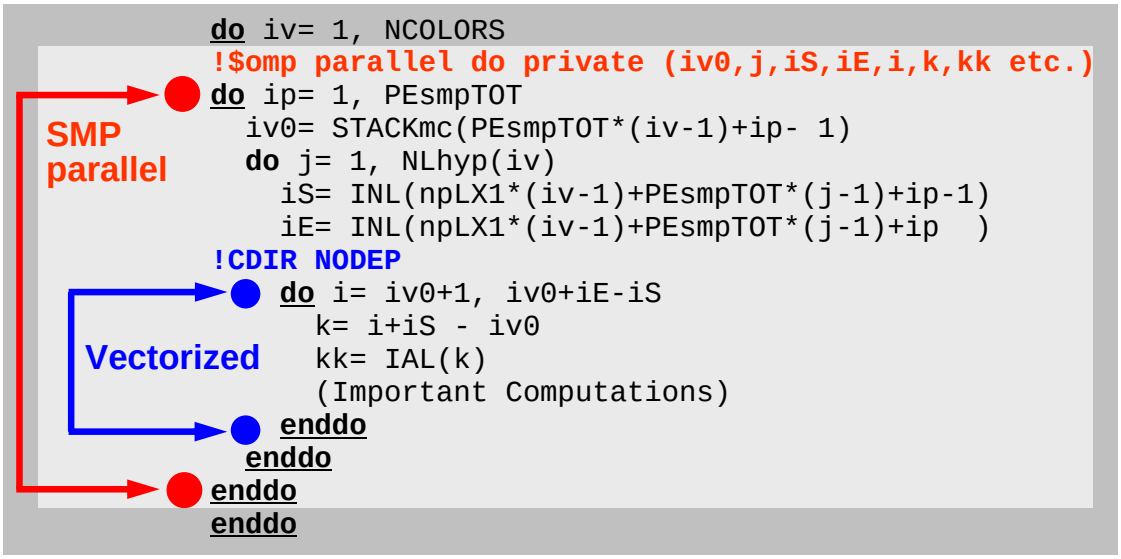


Fig. 3.7 Forward/backward substitution procedure during ILU(0)/IC(0) preconditioning by PDJDS/CM-RCM reordering (1) [81]

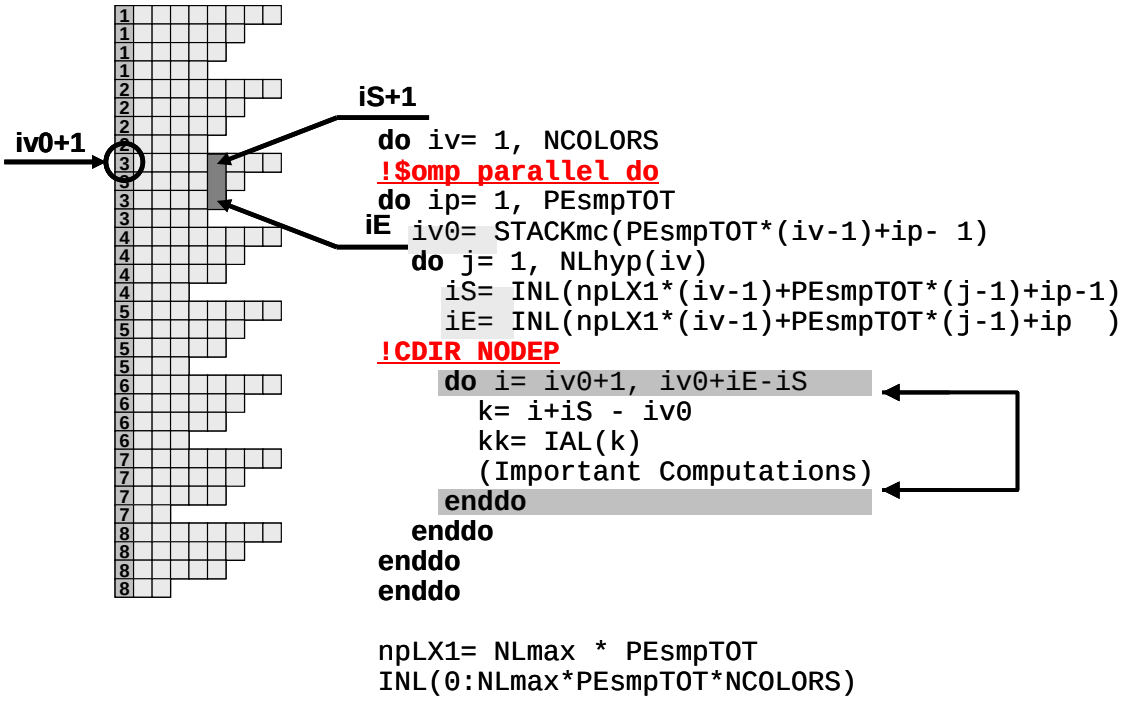


Fig. 3.8 Forward/backward substitution procedure during ILU(0)/IC(0) preconditioning by PDJDS/CM-RCM reordering (2) [81]

(a) Matrix-Vector Products

```
!$omp parallel do private(iS,iE,i,Xm0,Xm1,Xm2)
      do ip= 1, PEsmpTOT
        iS= STACKmcG(ip-1) + 1
        iE= STACKmcG(ip  )
!CDIR NODEP
        do i= iS, iE
          Xm2= X(3*i-2)
          Xm1= X(3*i-1)
          Xm0= X(3*i  )
          W(3*i-2,R)= B(3*i-2)-D(9*i-8)*Xm2-D(9*i-7)*Xm1-D(9*i-6)*Xm0
          W(3*i-1,R)= B(3*i-1)-D(9*i-5)*Xm2-D(9*i-4)*Xm1-D(9*i-3)*Xm0
          W(3*i  ,R)= B(3*i  )-D(9*i-2)*Xm2-D(9*i-1)*Xm1-D(9*i  )*Xm0
        enddo
      enddo
!$omp end parallel do
```

(b) Inner Dot Products

```
!$omp parallel do private(iS,iE,i)
!$omp&            reduction(+:RHO01,RHO02,RHO03)
      do ip= 1, PEsmpTOT
        iS= STACKmcG(ip-1) + 1
        iE= STACKmcG(ip  )
!CDIR NODEP
        do i= iS, iE
          RHO01= RHO01 + W(3*i-2,R)*W(3*i-2,Z)
          RHO02= RHO02 + W(3*i-1,R)*W(3*i-1,Z)
          RHO03= RHO03 + W(3*i  ,R)*W(3*i  ,Z)
        enddo
      enddo
!$omp end parallel do
```

(c) DAXPY

```
!$omp parallel do
!CDIR NODEP
      do i= 1, NP
        W(3*i-2,1)= W(3*i-2,2) + ALPHA*W(3*i-2,3)
        W(3*i-1,1)= W(3*i-1,2) + ALPHA*W(3*i-1,3)
        W(3*i  ,1)= W(3*i  ,2) + ALPHA*W(3*i  ,3)
        ....
      enddo
!$omp end parallel do
```

Fig. 3.9 Typical procedures in iterative methods with OpenMP directives and directives for vectorization of the Earth Simulator (a)Matrix-Vector Products, (b)Inner Dot Products, (c)DAXPY [81]

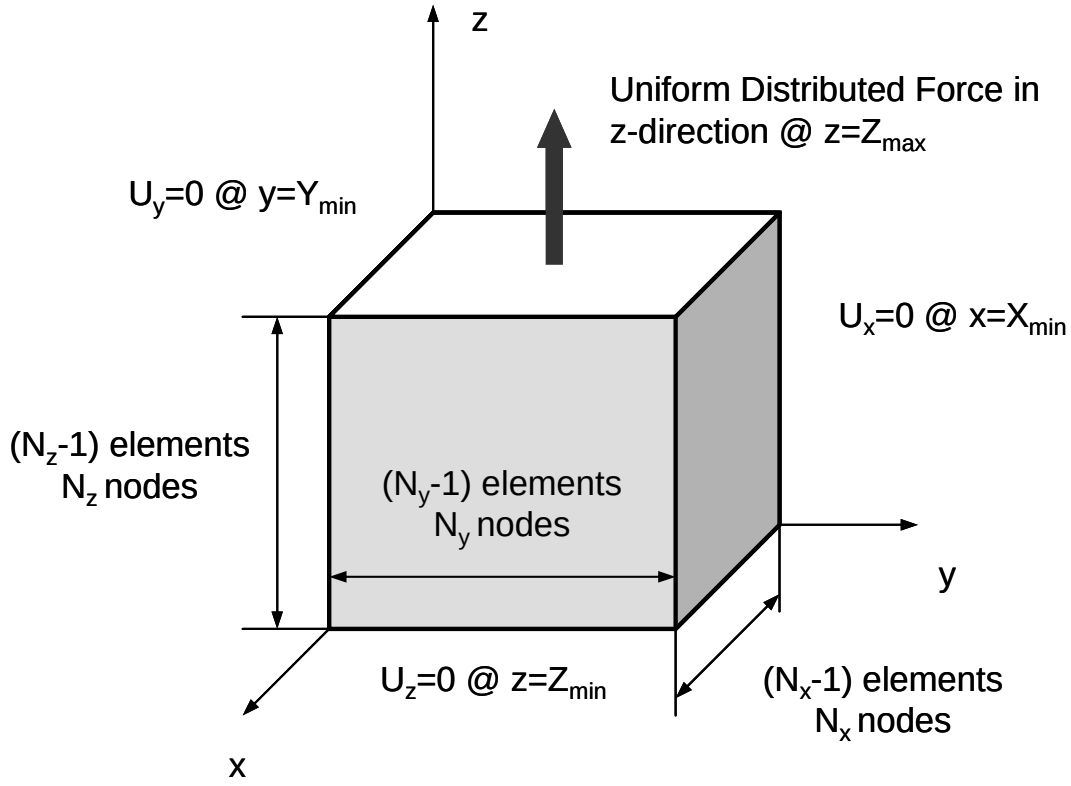
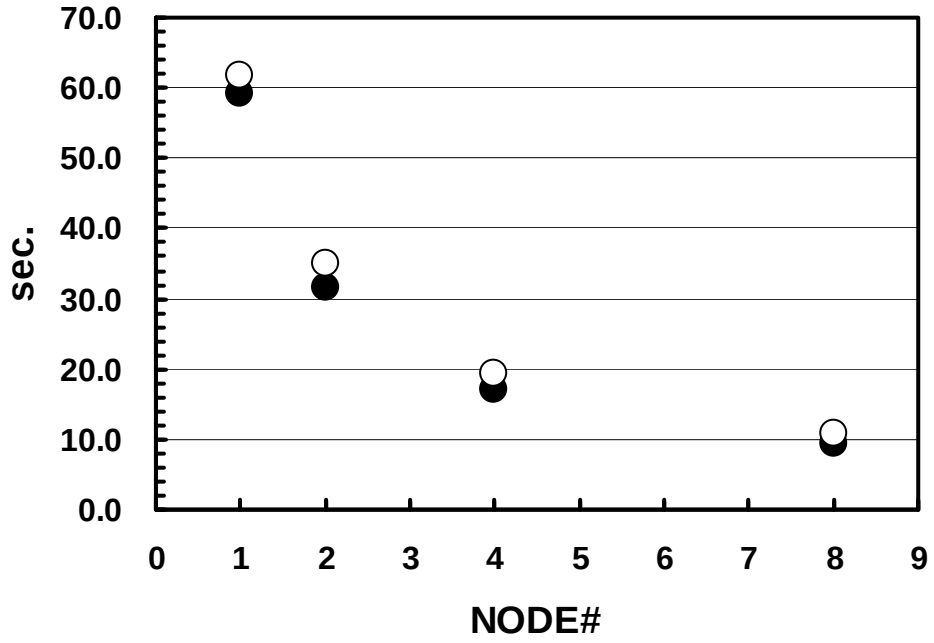


Fig. 3.10 Problem definition and boundary conditions for 3D solid mechanics example cases.

(a) Elapsed Time



(b) Parallel Speed-Up Ratio

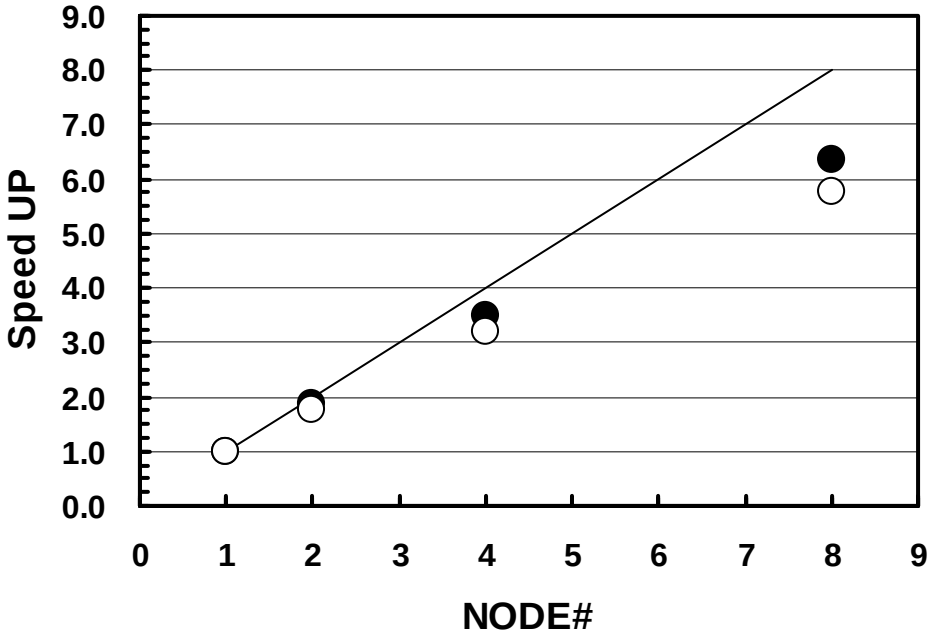
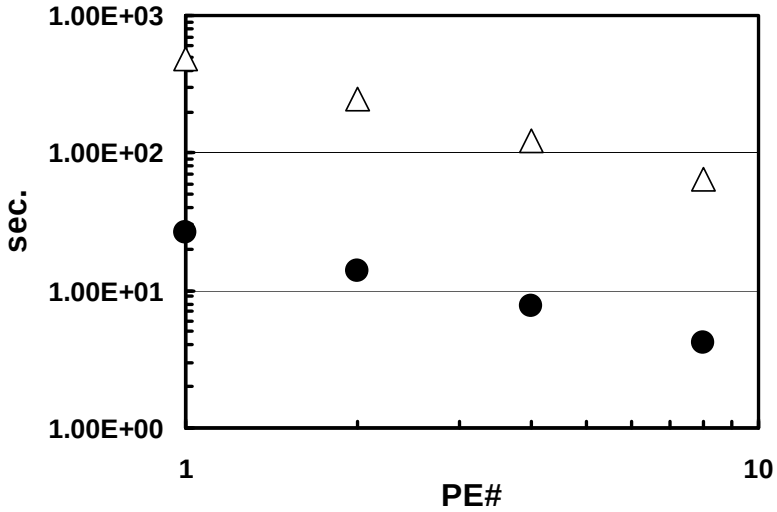
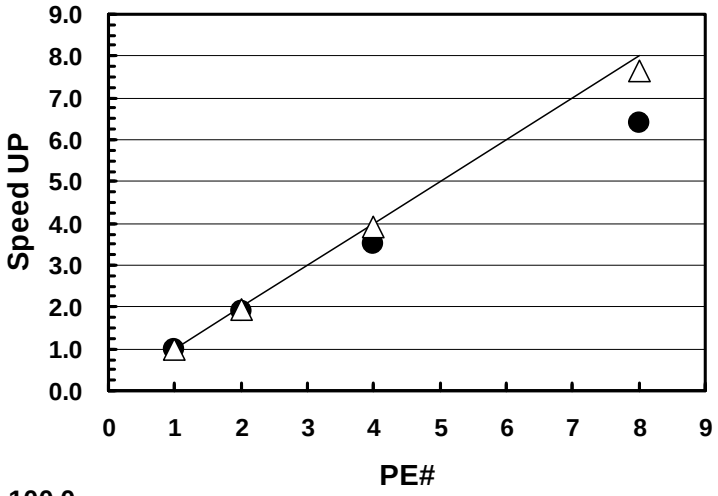


Fig. 3.11 Relationship between number of SMP nodes and the speedup for the 3D linear elastic problem in Fig.3.10 on the Earth Simulator with PDJDS/CM-RCM reordering. The total problem size is fixed at 3×128^3 (6,291,456) DOF. Speedup rate for 8 SMP nodes is is 6.36 (Flat MPI) and 5.78 (Hybrid). (BLACK Circles: Flat MPI, WHITE Circles: Hybrid).

(a) Elapsed Time



(b) Parallel Speed-Up Ratio



(c) GLOPS Rate

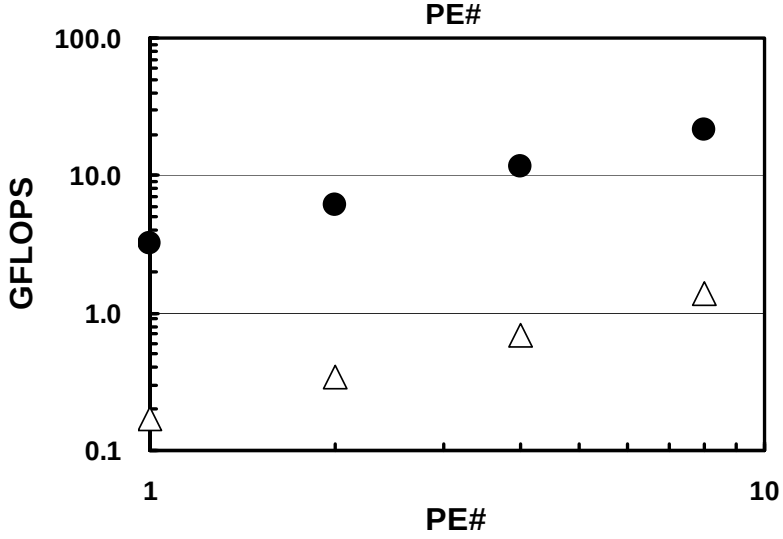
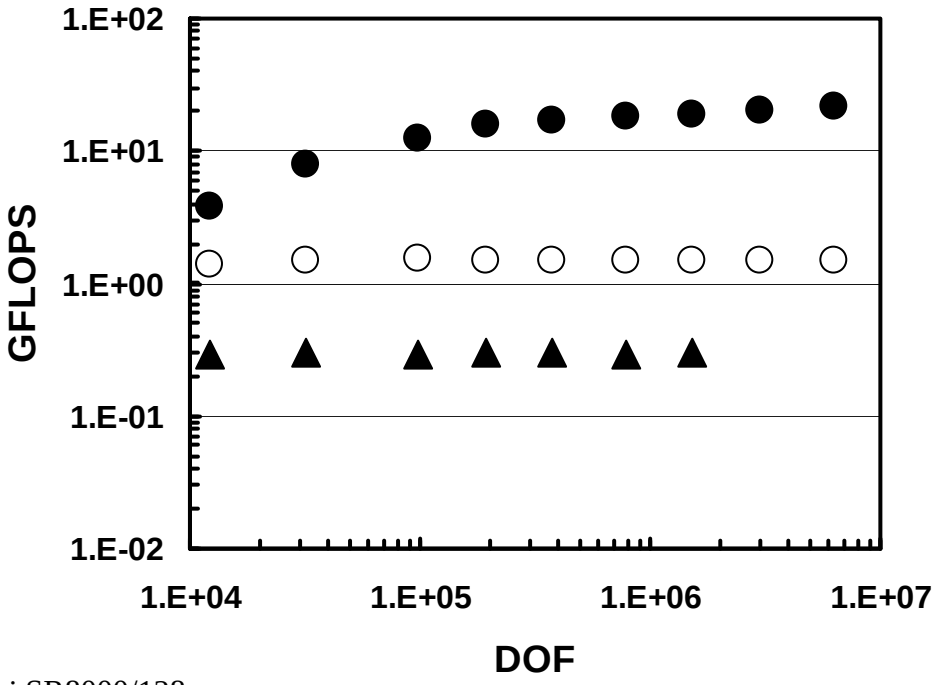


Fig. 3.12 Relationship between number of PEs and the parallel performance for the 3D linear elastic problem in Fig.3.10 on the Earth Simulator. (a) Elapsed time, (b) Parallel Speed –up ratio and (c) GFLOPS rate. The total problem size is fixed at 3×64^3 (786,432) DOF. (BLACK Circles: PDJDS/CM-RCM reordering, WHITE Triangles: Original GeoFEM solver without optimization).

(a) Earth Simulator



(b) Hitachi SR8000/128

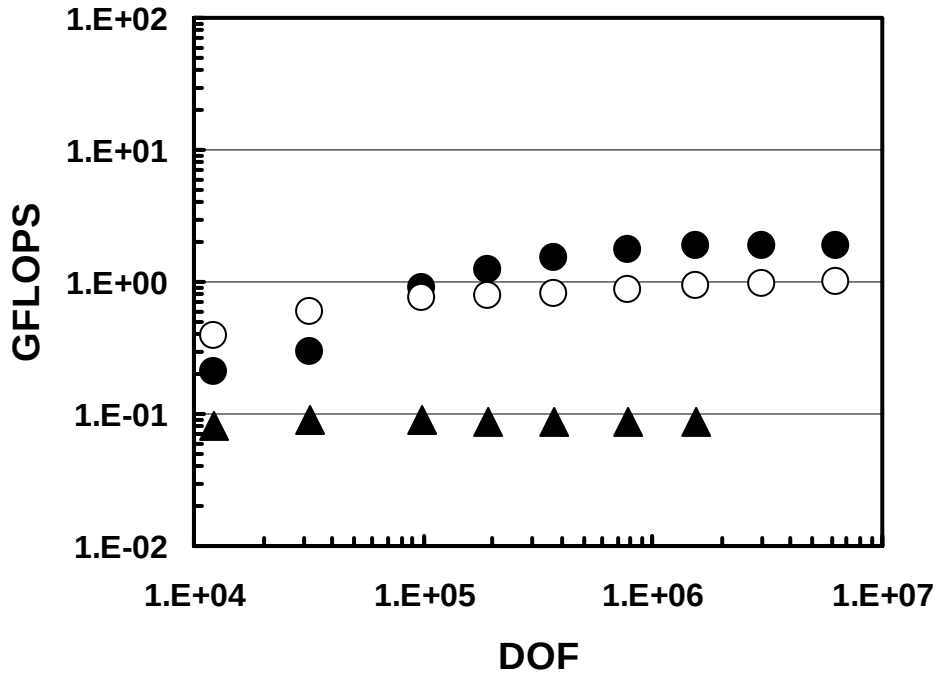


Fig. 3.13 Effect of coefficient matrix storage method and reordering for the 3D linear elastic problem in Fig.3.10 with various problem sizes on (a) Earth Simulaotr and (b) Hitachi SR8000/128 with 1 SMP node. The performance of the solver without reordering is very low due to synchronization overhead during forward/backward substitution for the IC factorization (BLACK Circles: PDJDS/CM-RCM, WHITE Circles: PDCRS/CM-RCM, BLACK Triangles: CRS no reordering).

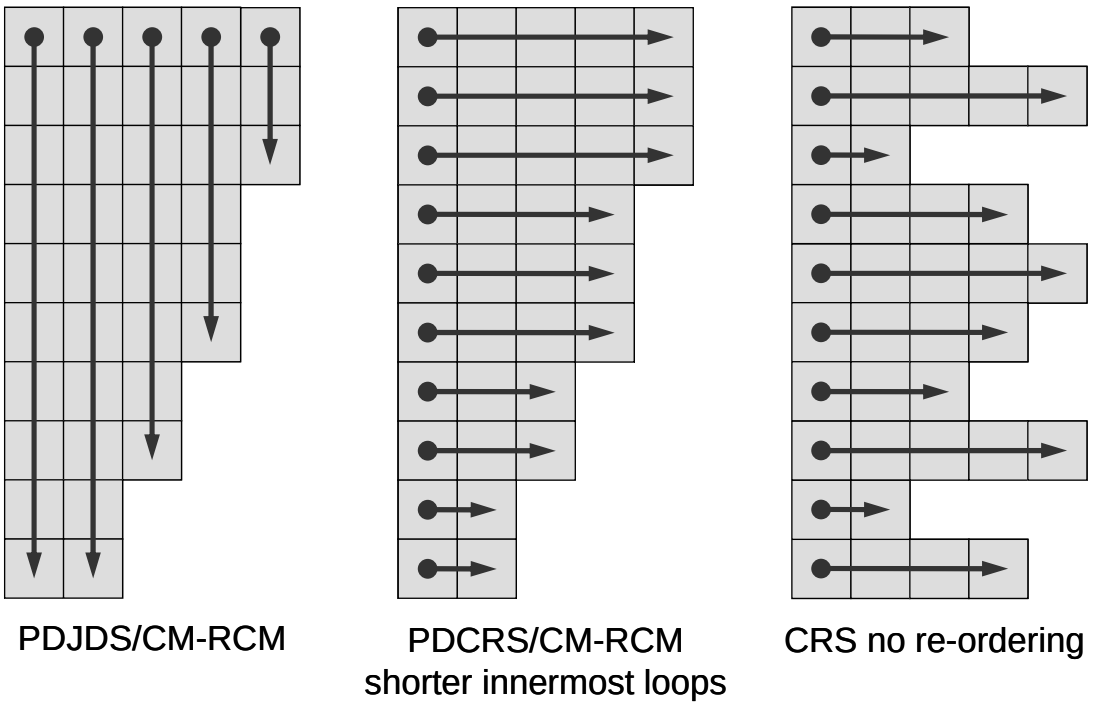


Fig. 3.14 Three types of matrix storage methods [81]

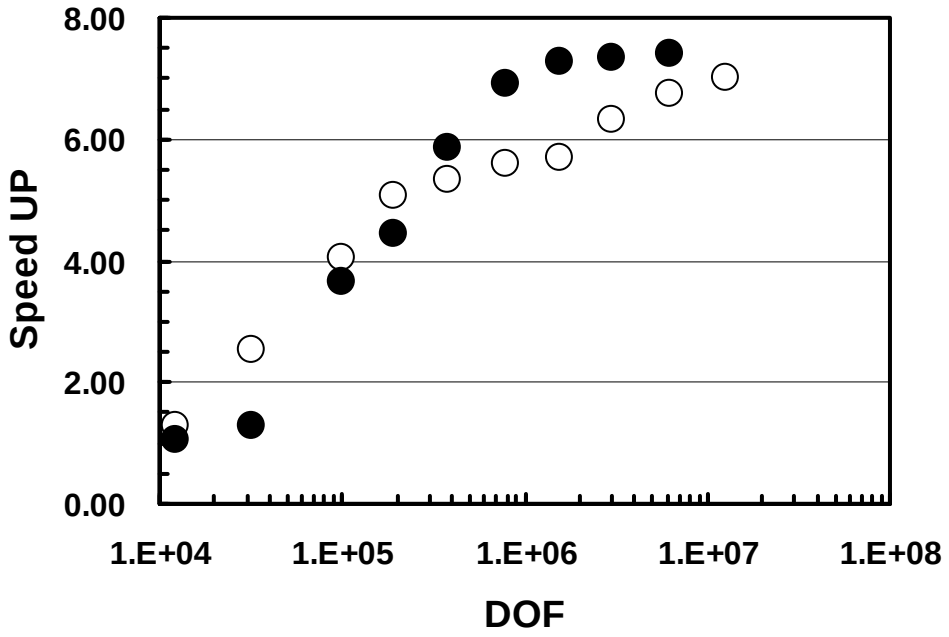


Fig. 3.15 Speed-up ratio inside SMP node for the 3D linear elastic problem in Fig.3.10 with various problem sizes on the Earth Simulator and the Hitachi SR8000/MPP with 1 SMP node with PDJDS/CM-RCM reordering. (WHITE Circles: Earth Simulator, BLACK Circles: Hitachi SR8000/128). Speed-up ratio for sufficiently large problem is 7.01 (Earth Simulator) and 7.40 (Hitachi SR8000/128).

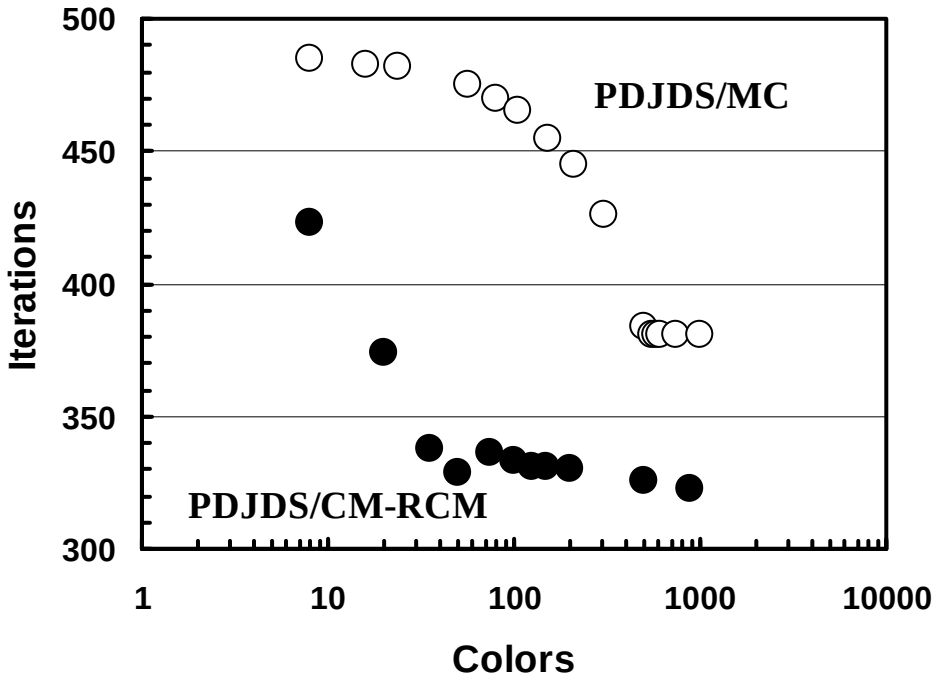
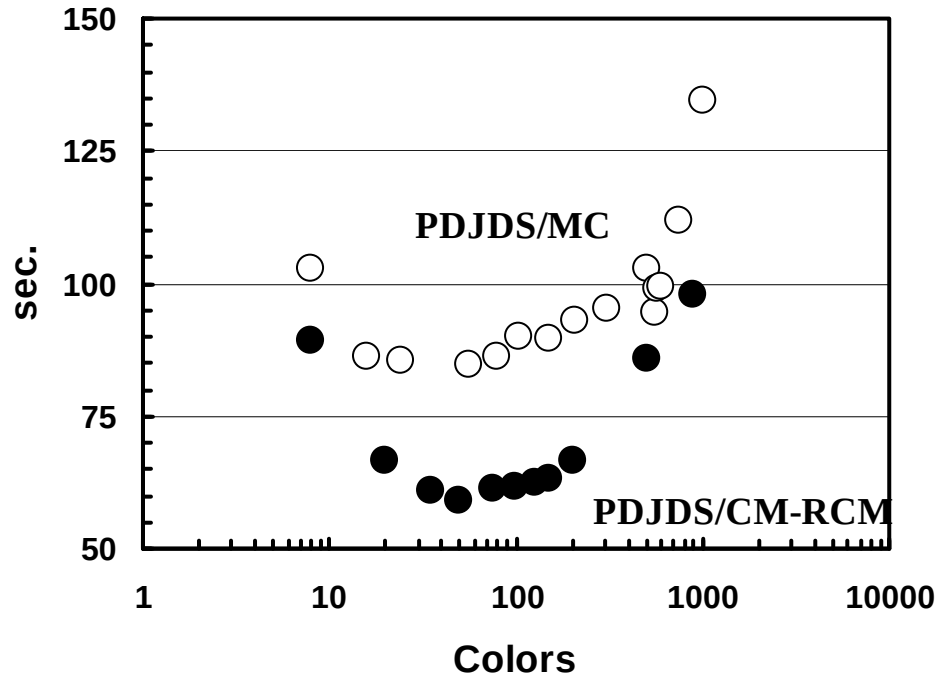


Fig. 3.16 Effect of reordering method and number of colors (Number of iterations for convergence). 3D linear elastic problem in Fig.3.10 with 3×128^3=6,291,456 DOF on the Earth Simulator and Hitachi SR8000/128 with 1 SMP node. Number of iterations for convergence. (BLACK Circles: PDJDS/CM-RCM, WHITE Circles: PDJDS/MC). PDJDS/CM-RCM provides better performance. In many colors, number of iterations for convergence is smaller but performance is worse due to smaller loop length and overhead.

(a) Earth Simulator



(b) SR8000/128

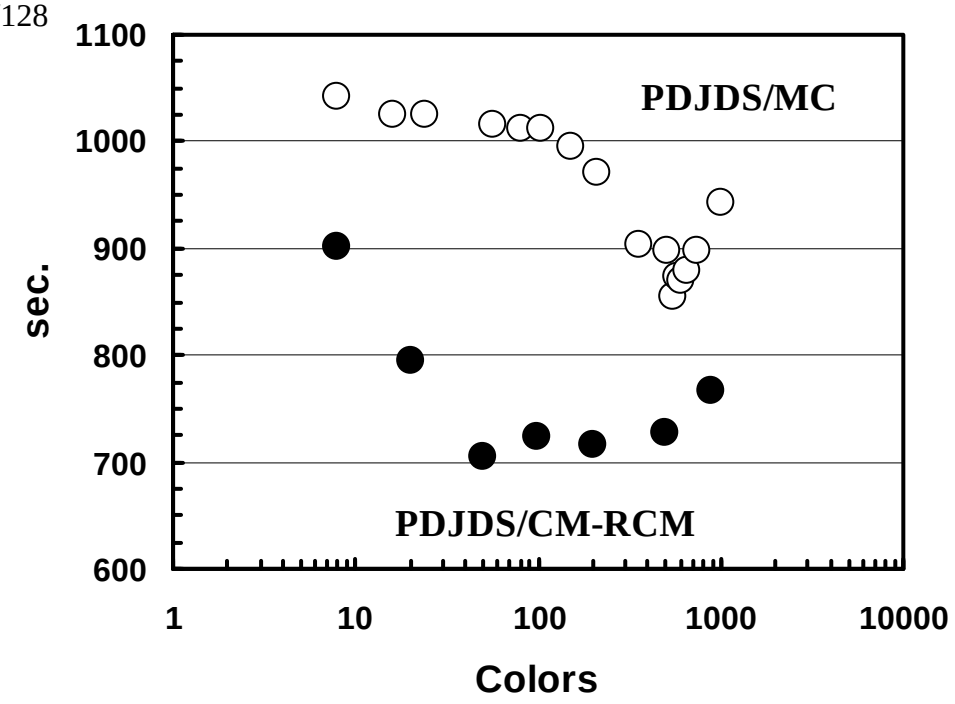
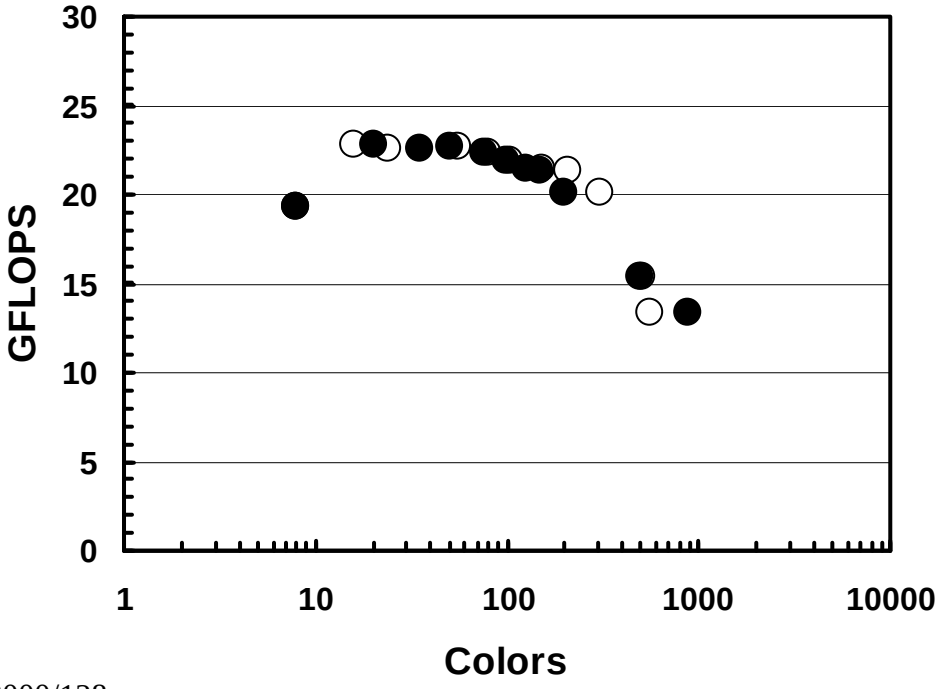


Fig. 3.17 Effect of reordering method and number of colors (CPU time, sec.). 3D linear elastic problem in Fig.3.10 with 3×128^3=6,291,456 DOF on (a) Earth Simulaotr and (b) Hitachi SR8000/128 with 1 SMP node. CPU time. (BLACK Circles: PDJDS/CM-RCM, WHITE Circles: PDJDS/MC). PDJDS/CM-RCM provides better performance. In many colors, number of iterations for convergence is smaller but performance is worse due to smaller loop length and overhead.

(a) Earth Simulator



(b) SR8000/128

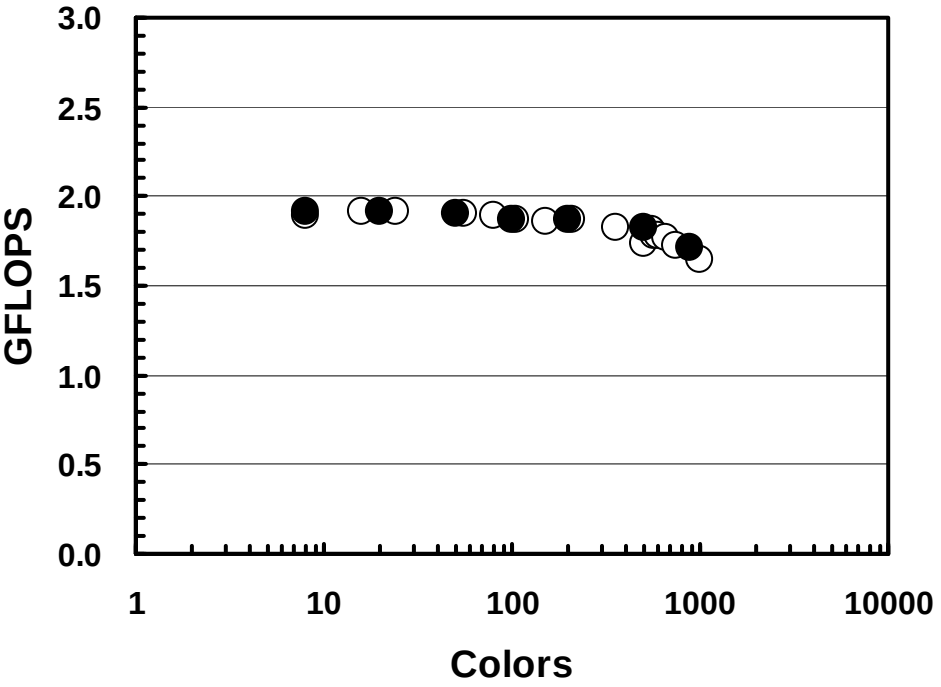


Fig. 3.18 Effect of reordering method and number of colors (GFLOPS rate). 3D linear elastic problem in Fig.3.10 with 3×128^3=6,291,456 DOF on (a) Earth Simulaotr and (b) Hitachi SR8000/128 with 1 SMP node. GFLOPS rate. (BLACK Circles: PDJDS/CM-RCM, WHITE Circles: PDJDS/MC). PDJDS/CM-RCM provides better performance. In many colors, number of iterations for convergence is smaller but performance is worse due to smaller loop length and overhead.

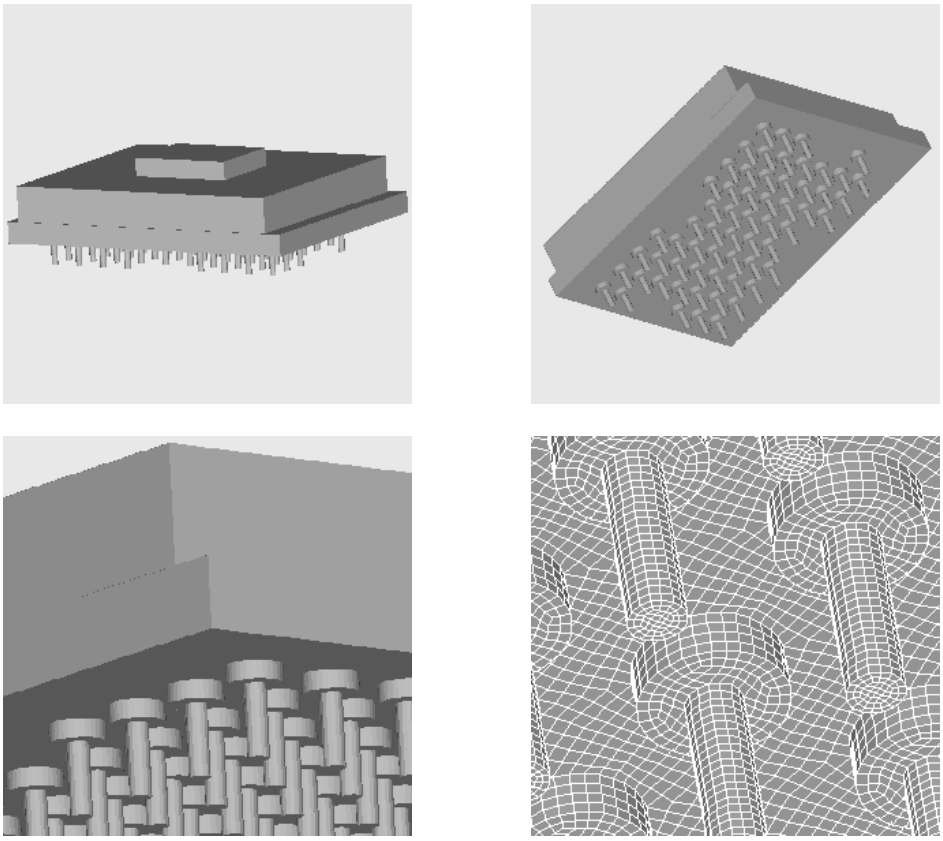


Fig. 3.19 Micro PGA model for 3D linear elastic analysis. 61 pins, 956,128 elements, 1,012,354 nodes (3,037,062 DOF) [143]

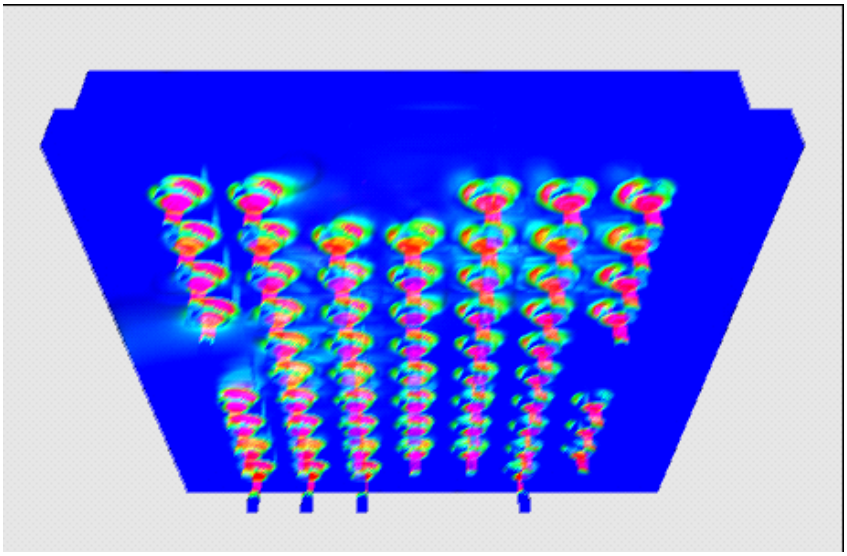


Fig. 3.20 Results of 3D linear elastic analysis in Micro PGA model. Stress intensity distribution visualized by GeoFEM's volume rendering [15].

Table 3.1 Effect of reordering method and number of colors. 3D linear elastic analysis of Micro PGA model (Fig.3.19) with 956,128 elements, 1,012,354 nodes and 3,037,062 DOF on the (a) Earth Simulator and (b)Hitachi SR8000/128 with 1 SMP node. Number of iterations for convergence is smaller in PDJDS/CM-RCM but CPU time is longer because the number of color is large and performance is worse due to smaller loop length and overhead.

(a) Earth Simulator

METHOD	MC	MC	MC	MC	MC	CM-RCM(*1)	CM-RCM(*2)
COLOR#	108	158	204	306	1012	2381	1773
Iterations	474	465	447	436	434	377	371
sec.	49.7	53.2	54.2	60.4	120.2	206.4	159.3
GFLOPS	18.7	17.2	16.2	14.2	7.09	3.59	4.14
Performance(*3)	29.3%	26.8%	25.3%	22.2%	11.1%	5.60%	7.14%

(b) Hitachi SR800/128

METHOD	MC	MC	MC	MC	MC	CM-RCM(*1)	CM-RCM(*2)
COLOR#	108	158	204	306	1012	2381	1773
Iterations	474	465	447	436	434	377	371
sec.	493	495	480	490	676	996	644
GFLOPS	1.89	1.84	1.83	1.75	1.26	0.74	1.13
Performance(*4)	23.6%	23.0%	22.8%	21.8%	15.8%	9.3%	14.1%

(1*): No rooting method is applied.
(2*): Rooting method by Gibbs [81] is applied before RCM ordering
(3*): Performance compared to the peak performance (64GFLOPS/SMP node).
(4*): Performance compared to the peak performance (8GFLOPS/SMP node).

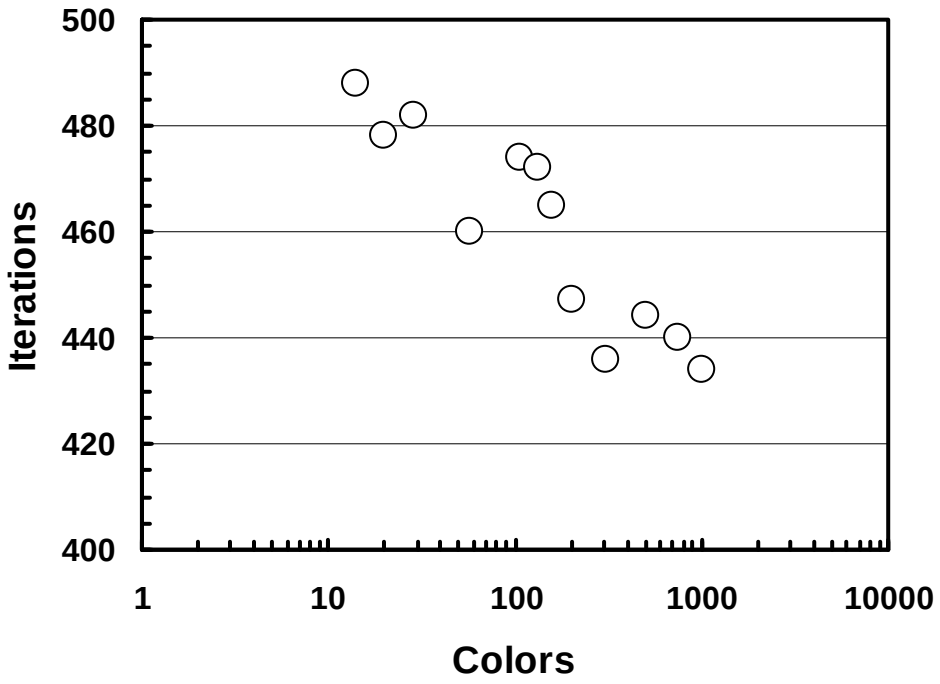
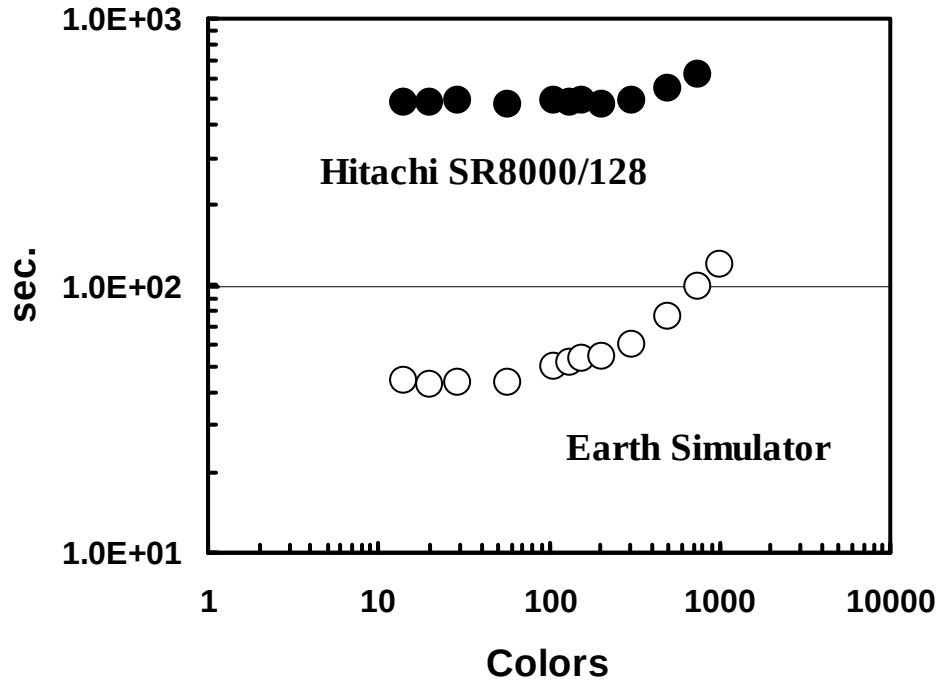


Fig. 3.21 Effect of reordering method and number of colors (Number of iterations for convergence). 3D linear elastic analysis of Micro PGA model (Fig.3.19) with 956,128 elements, 1,012,354 nodes and 3,037,062 DOF on the Earth Simulator and Hitachi SR8000/128 with 1 SMP node. Number of iterations with PDJDS/MC.

(a) Elapsed Time



(b) GFLOPS Rate

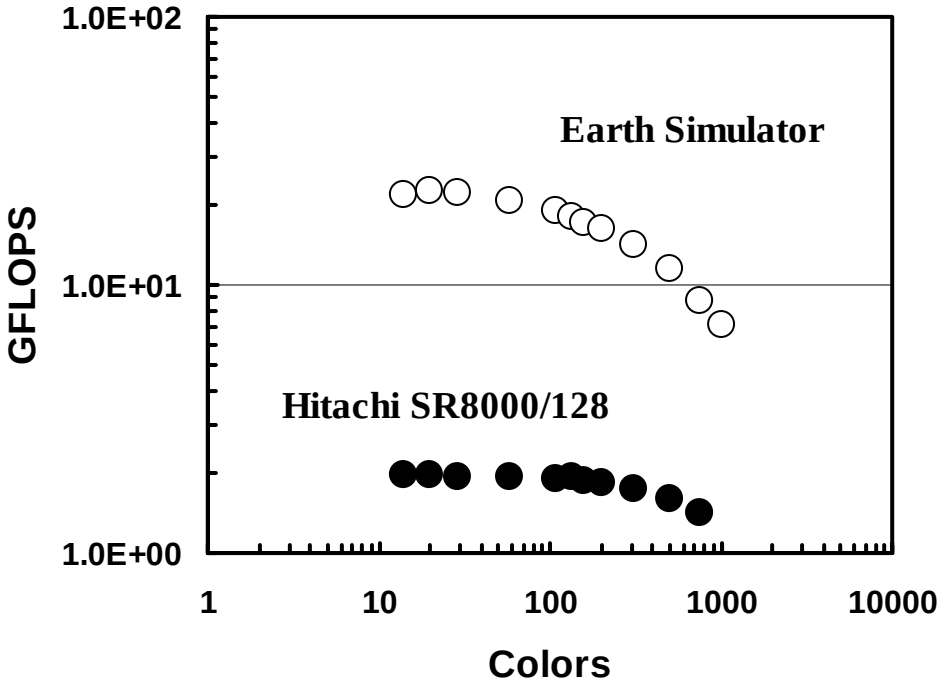


Fig. 3.22 Effect of reordering method and number of colors (Number of iterations for convergence). 3D linear elastic analysis of Micro PGA model (Fig.3.19) with 956,128 elements, 1,012,354 nodes and 3,037,062 DOF on the Earth Simulator and Hitachi SR8000/128 with 1 SMP node. (a) Elapsed time and (b) GFLOPS rate with PDJDS/MC. (WHITE Circles: Earth Simulator, BLACK Circles: Hitachi SR8000/128).

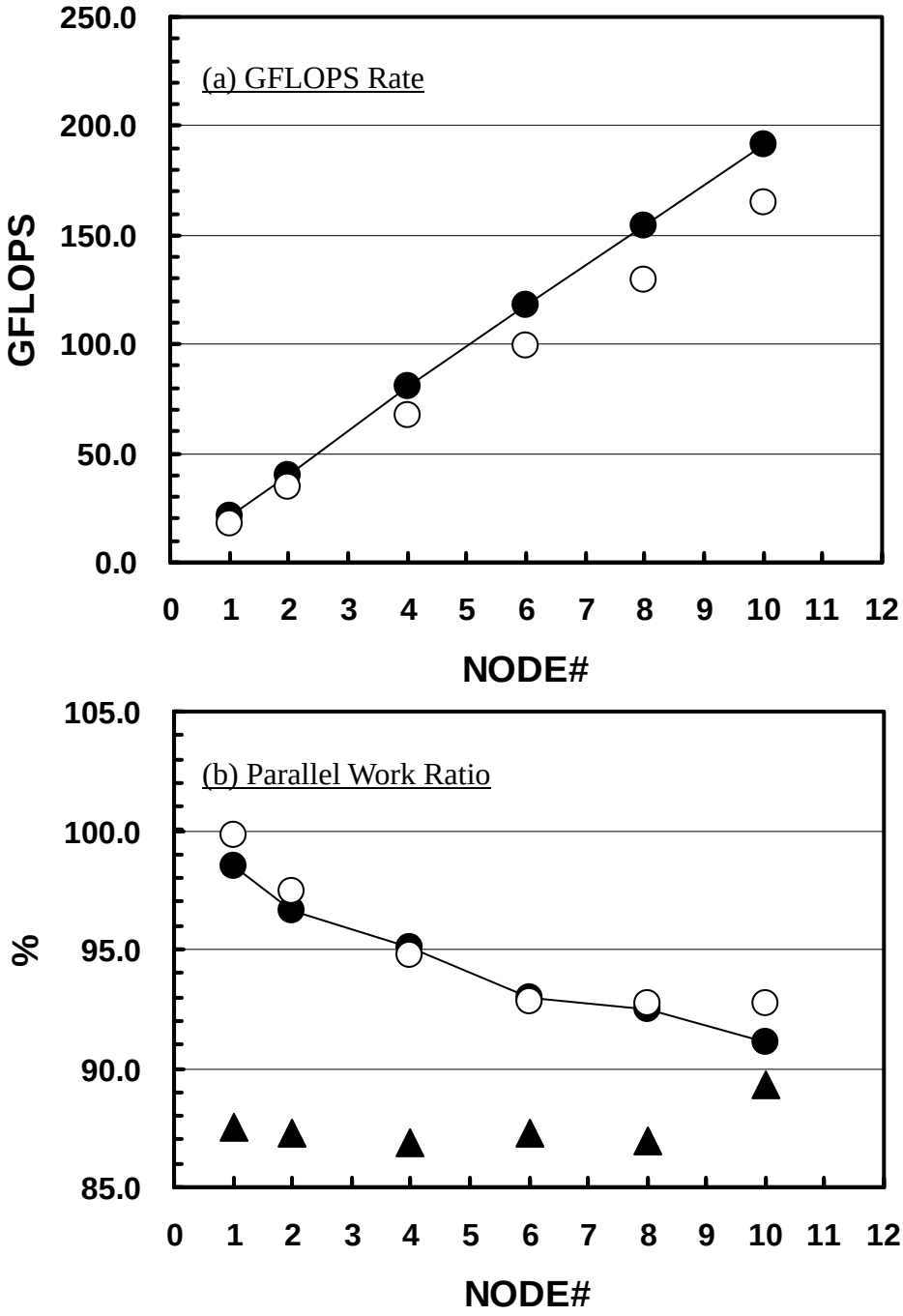


Fig. 3.23 Problem size and parallel performance on the Earth Simulator for the 3D linear elastic problem in Fig.3.10 using between 1 and 10 SMP nodes. (a)GFLOPS rate and (b)Parallel work ratio. Problem size/SMP node is fixed as 786,432 DOF (3×64^3). Largest case is 7,864,320 DOF on 10 SMP nodes (80 PEs). Maximum performance is 192 (Flat MPI) and 165 (Hybrid) GFLOPS (Peak performance= 640 GFLOPS). (BLACK Circles: Flat MPI, WHITE Circles: Hybrid, BLACK Triangles: Hybrid/Flat MPI Performance Ratio based on Elapsed Time of Flat MPI). PDJDS/CM-RCM reordering.

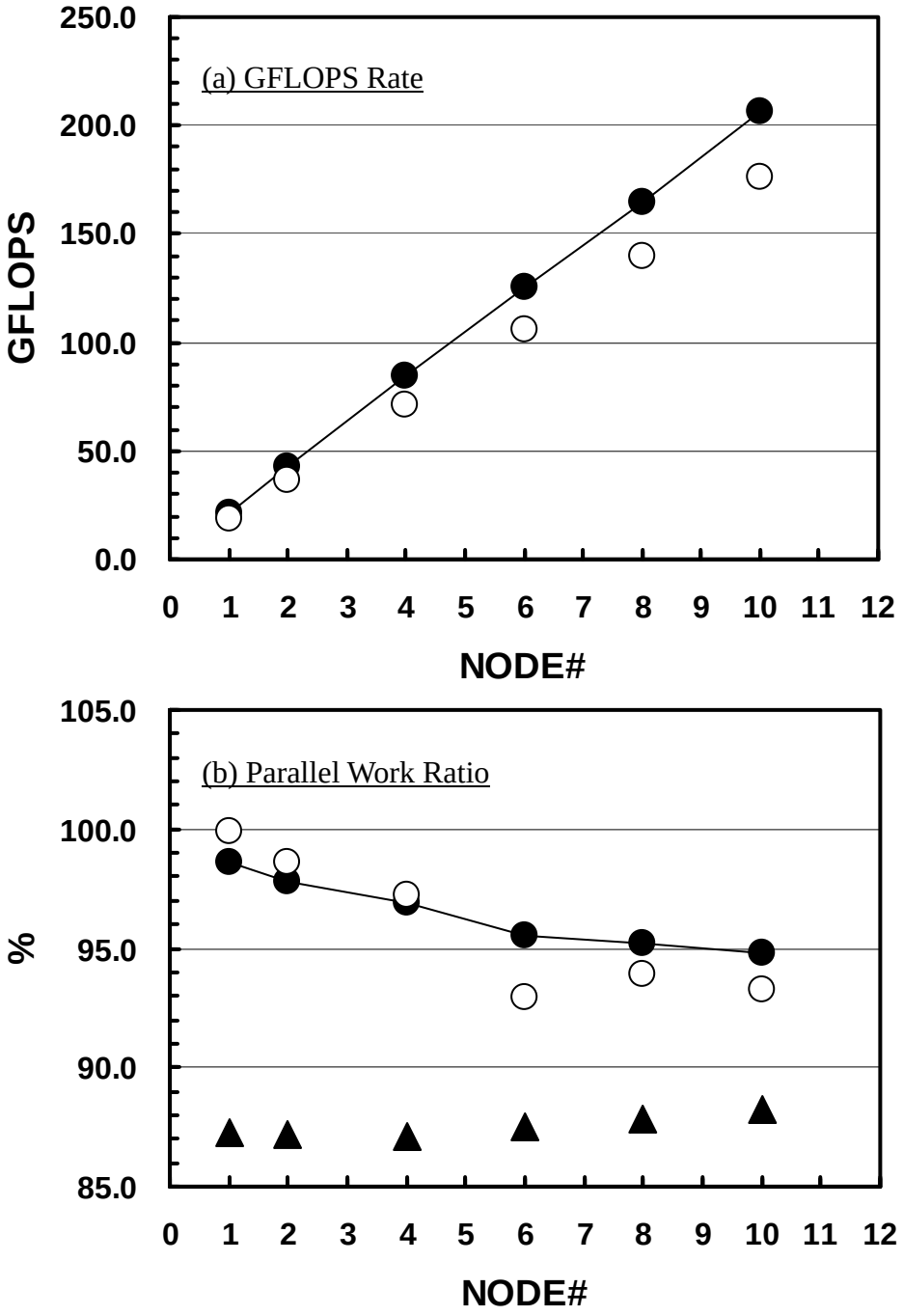


Fig. 3.24 Problem size and parallel performance on the Earth Simulator for the 3D linear elastic problem in Fig.3.10 between 1 and 10 SMP nodes. (a)GFLOPS rate and (b)Parallel work ratio. Problem size/SMP node is fixed as 1,536,000 DOF (3×80^3). Largest case is 15,360,000 DOF on 10 SMP nodes (80 PEs). Maximum performance is 206 (Flat MPI) and 176 (Hybrid) GFLOPS (Peak performance= 640 GFLOPS). (BLACK Circles: Flat MPI, WHITE Circles: Hybrid, BLACK Triangles: Hybrid/Flat MPI Performance Ratio based on Elapsed Time of Flat MPI). PDJDS/CM-RCM reordering.

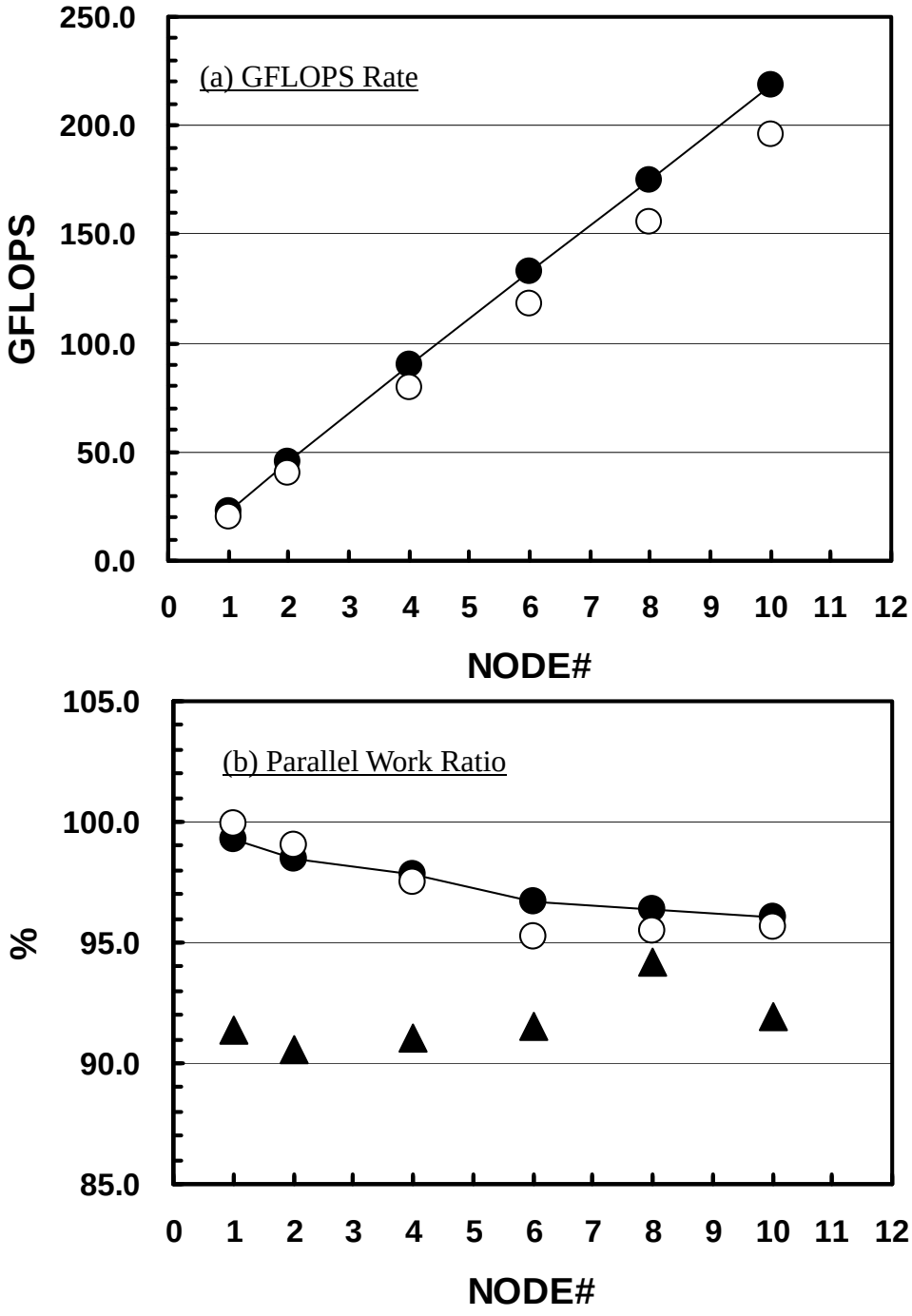


Fig. 3.25 Problem size and parallel performance on the Earth Simulator for the 3D linear elastic problem in Fig.3.10 using between 1 and 10 SMP nodes. (a)GFLOPS rate and (b)Parallel work ratio. Problem size/SMP node is fixed as 3,000,000 DOF (3×100^3). Largest case is 30,000,000 DOF on 10 SMP nodes (80 PEs). Maximum performance is 219 (Flat MPI) and 196 (Hybrid) GFLOPS (Peak performance= 640 GFLOPS). (BLACK Circles: Flat MPI, WHITE Circles: Hybrid, BLACK Triangles: Hybrid/Flat MPI Performance Ratio based on Elapsed Time of Flat MPI). PDJDS/CM-RCM reordering.

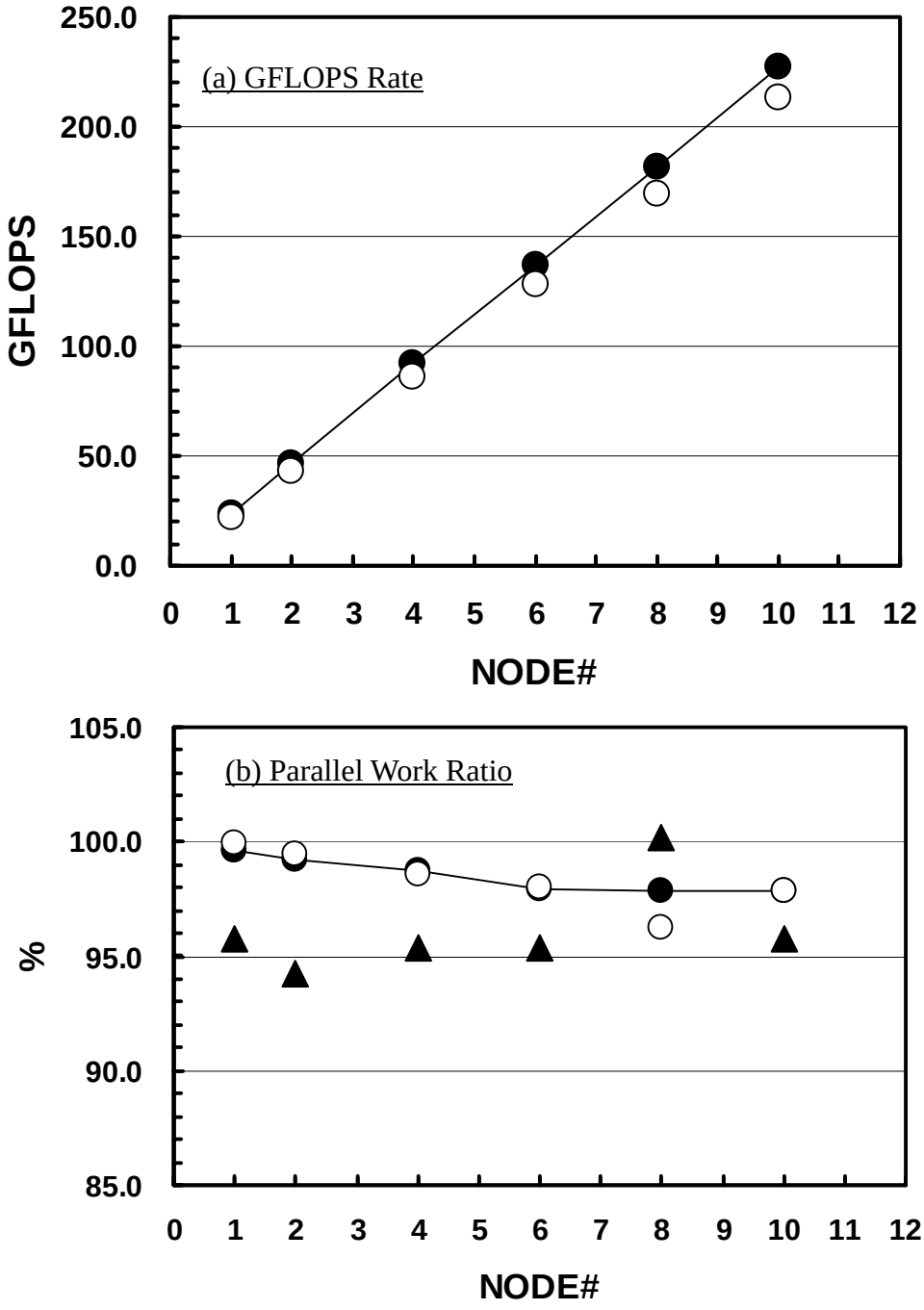


Fig. 3.26 Problem size and parallel performance on the Earth Simulator for the 3D linear elastic problem in Fig.3.10 using between 1 and 10 SMP nodes. (a)GFLOPS rate and (b)Parallel work ratio. Problem size/SMP node is fixed as 6,291,456 DOF (3×128^3). Largest case is 62,914,560 DOF on 10 SMP nodes (80 PEs). Maximum performance is 227 (Flat MPI) and 213 (Hybrid) GFLOPS (Peak performance= 640 GFLOPS). (BLACK Circles: Flat MPI, WHITE Circles: Hybrid, BLACK Triangles: Hybrid/Flat MPI Performance Ratio based on Elapsed Time of Flat MPI). PDJDS/CM-RCM reordering.

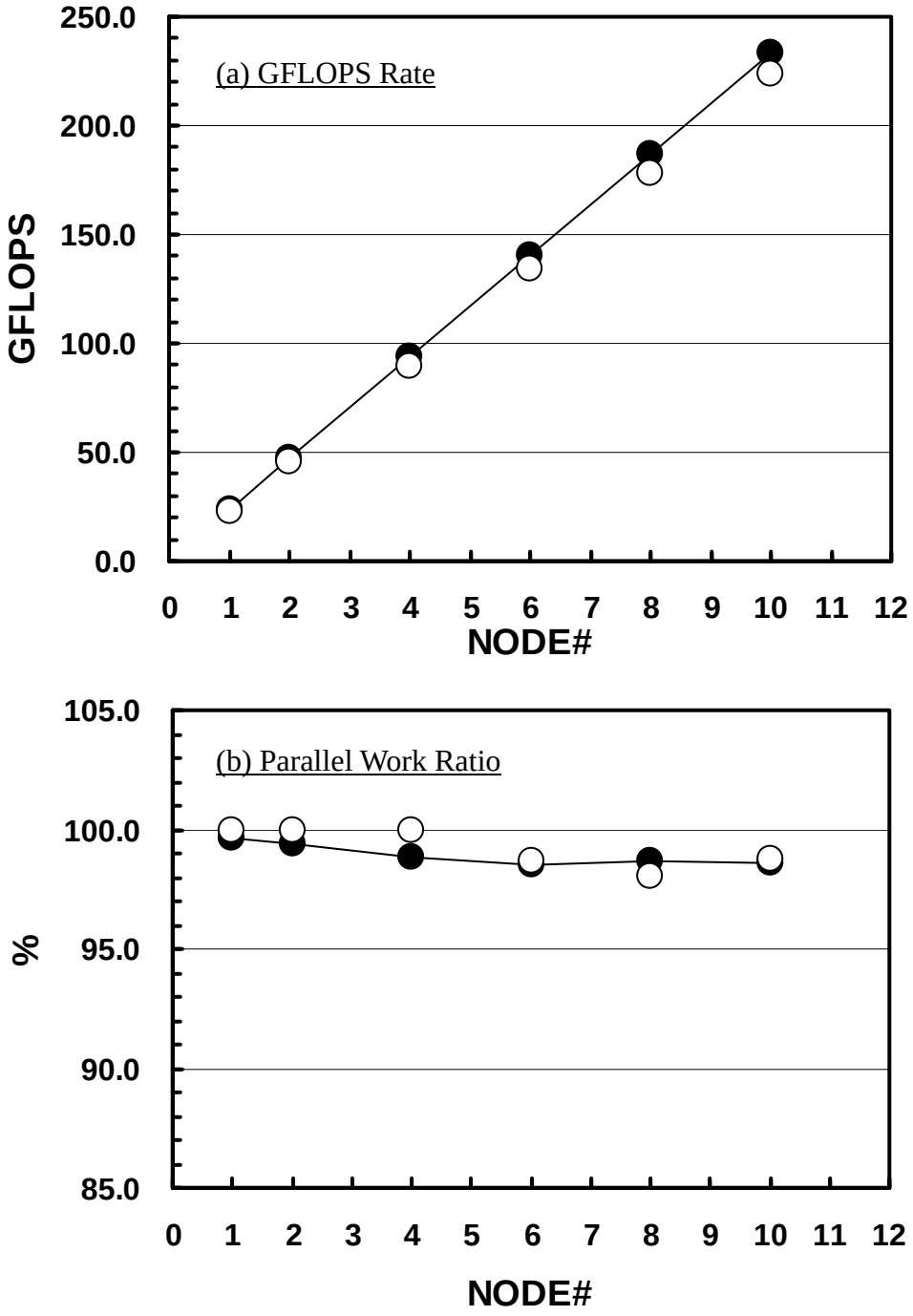
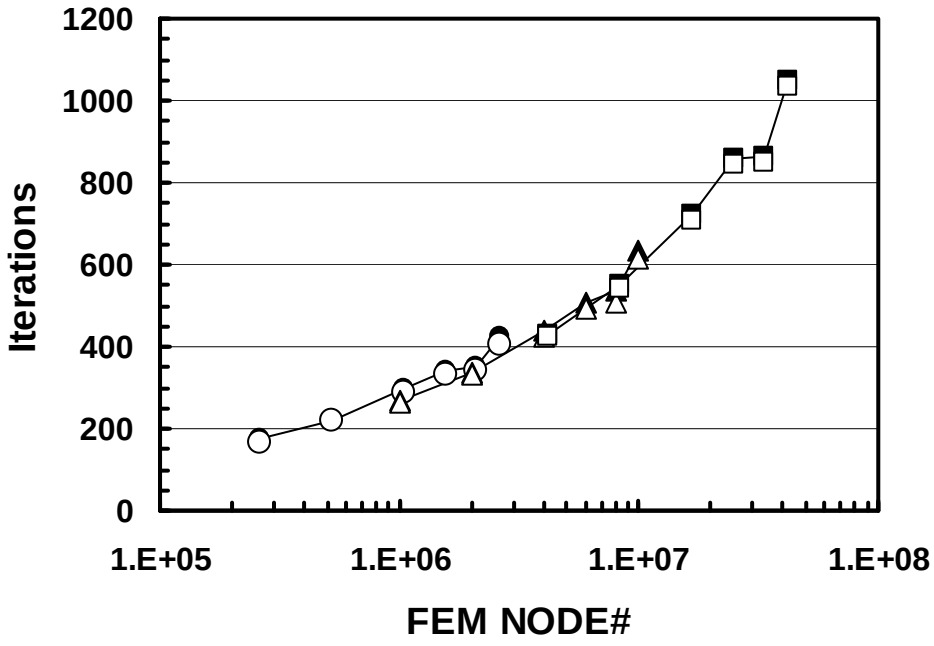


Fig. 3.27 Problem size and parallel performance on the Earth Simulator for the 3D linear elastic problem in Fig.3.10 using between 1 and 10 SMP nodes. (a)GFLOPS rate and (b)Parallel work ratio. Problem size/SMP node is fixed as 12,582,912 DOF (3×256×128×128). Largest case is 125,829,120 DOF on 10 SMP nodes (80 PEs). Maximum performance is 233 (Flat MPI) and 224 (Hybrid) GFLOPS (Peak performance= 640 GFLOPS). (BLACK Circles: Flat MPI, WHITE Circles: Hybrid, BLACK Triangles: Hybrid/Flat MPI Performance Ratio based on Elapsed Time of Flat MPI). PDJDS/CM-RCM reordering.

(a) Iterations for Convergence



(b) Performance Ratio to the Peak

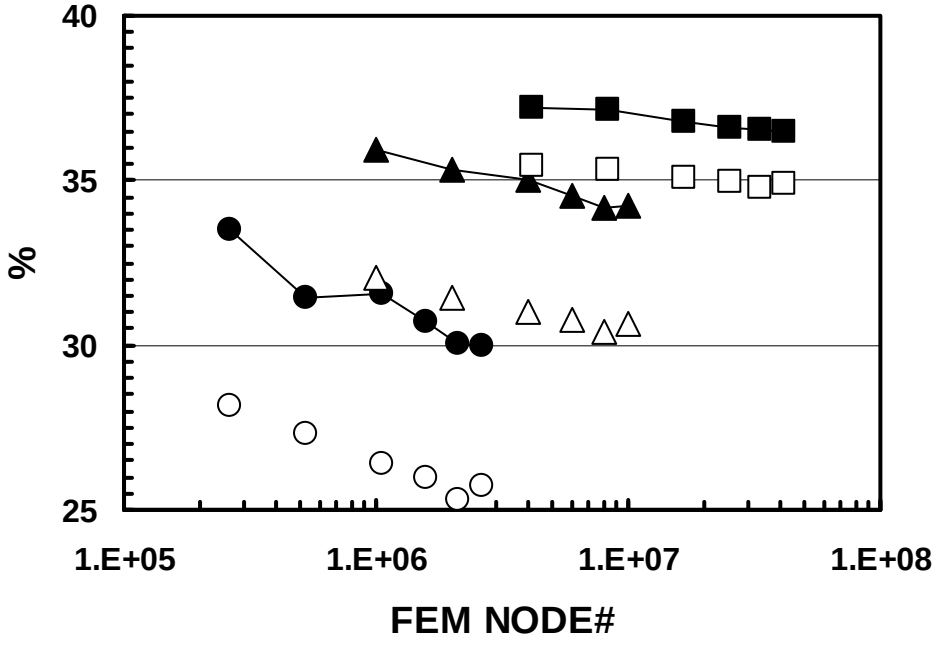


Fig. 3.28 Problem size and parallel performance on the Earth Simulator for the 3D linear elastic problem in Fig.3.10 using between 1 and 10 SMP nodes. (a) Iterations for convergence and (b) Performance ratio to the peak. Problem size/PE is fixed. (Circles: 3×64^3 DOF/SMP node, Triangles: 3×100^3 DOF/SMP node, Squares: $3\times256\times128\times128$ DOF/SMP node). (BLACK: Flat MPI, WHITE: Hybrid). PDJDS/CM-RCM reordering.

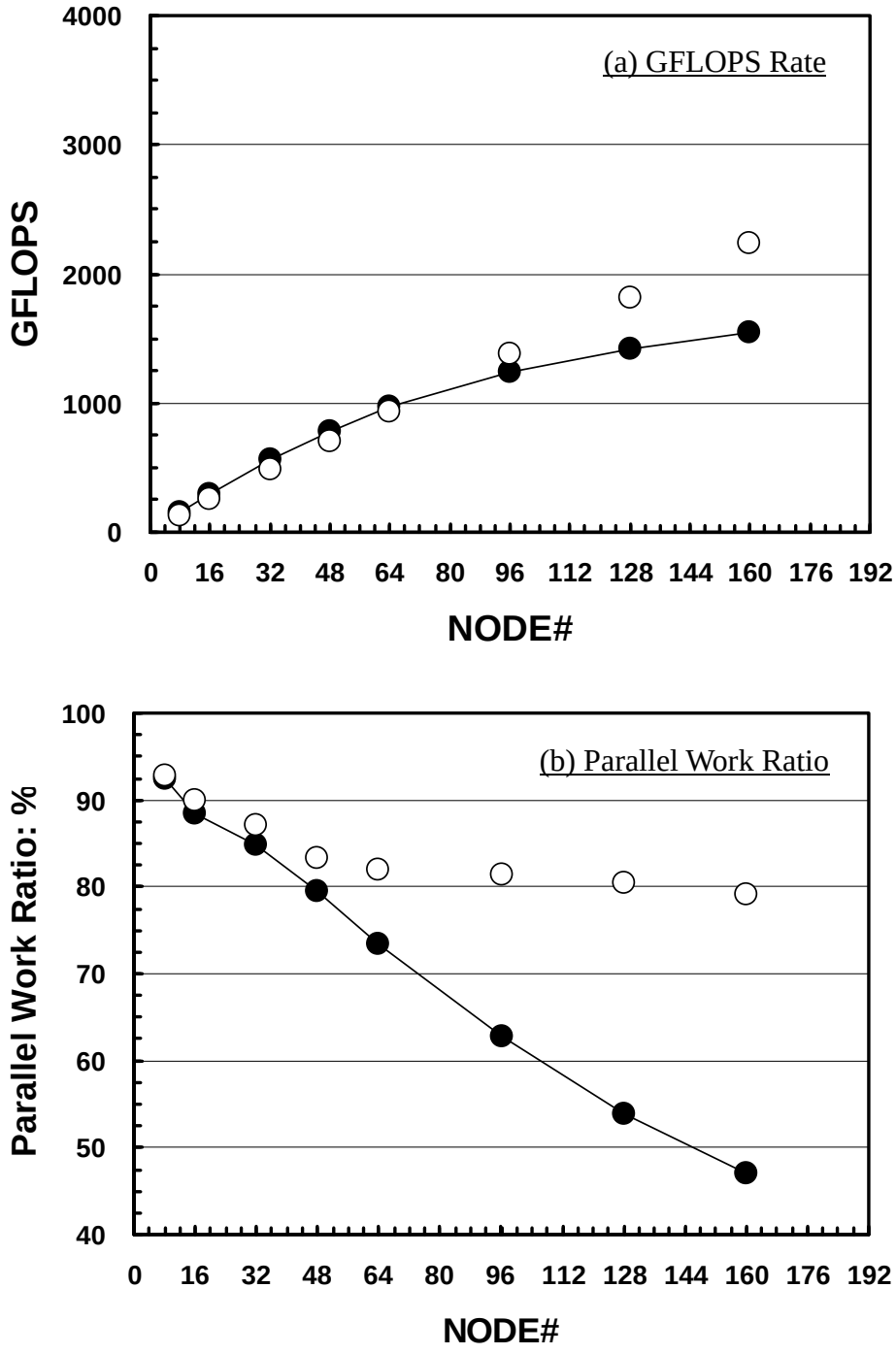


Fig. 3.29 Problem size and parallel performance on the Earth Simulator for the 3D linear elastic problem in Fig.3.10 using between 8 and 160 SMP nodes. (a)GFLOPS rate and (b)Parallel work ratio. Problem size/PE is fixed as 786,432 DOF (3×64^3). Largest case is 125,829,120 DOF on 160 SMP nodes (1280 PEs). Maximum performance is 1.55 (Flat MPI) and 2.23 (Hybrid) TFLOPS (Peak performance= 10.24 TFLOPS). (BLACK Circles: Flat MPI, WHITE Circles: Hybrid). PDJDS/CM-RCM reordering.

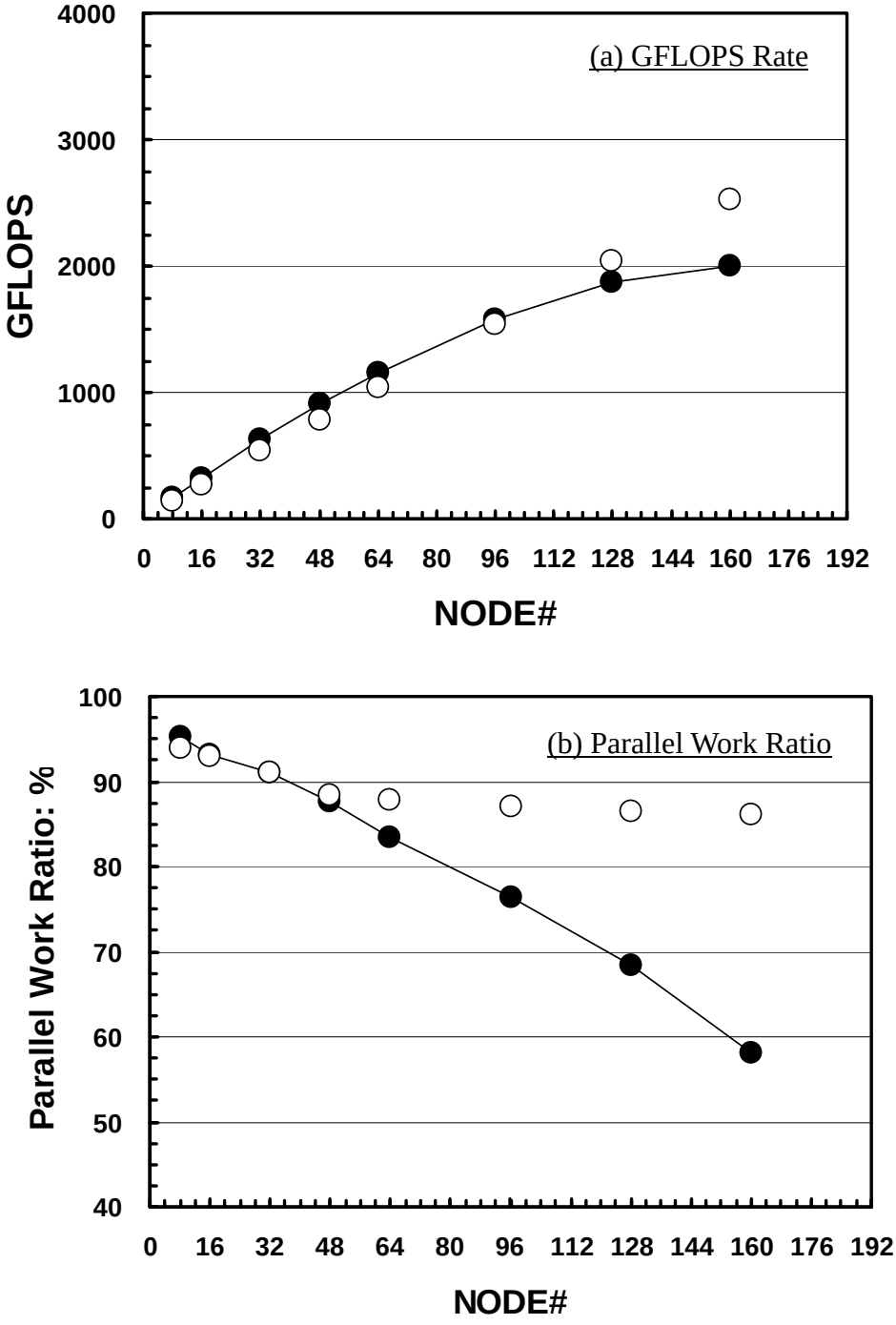


Fig. 3.30 Problem size and parallel performance on the Earth Simulator for the 3D linear elastic problem in Fig.3.10 using between 8 and 160 SMP nodes. (a)GFLOPS rate and (b)Parallel work ratio. Problem size/SMP node is fixed as 1,536,000 DOF (3×80^3). Largest case is 245,760,000 DOF on 160 SMP nodes (1280 PEs). Maximum performance is 2.00 (Flat MPI) and 2.53 (Hybrid) TFLOPS (Peak performance= 10.24 TFLOPS). (BLACK Circles: Flat MPI, WHITE Circles: Hybrid). PDJDS/CM-RCM reordering.

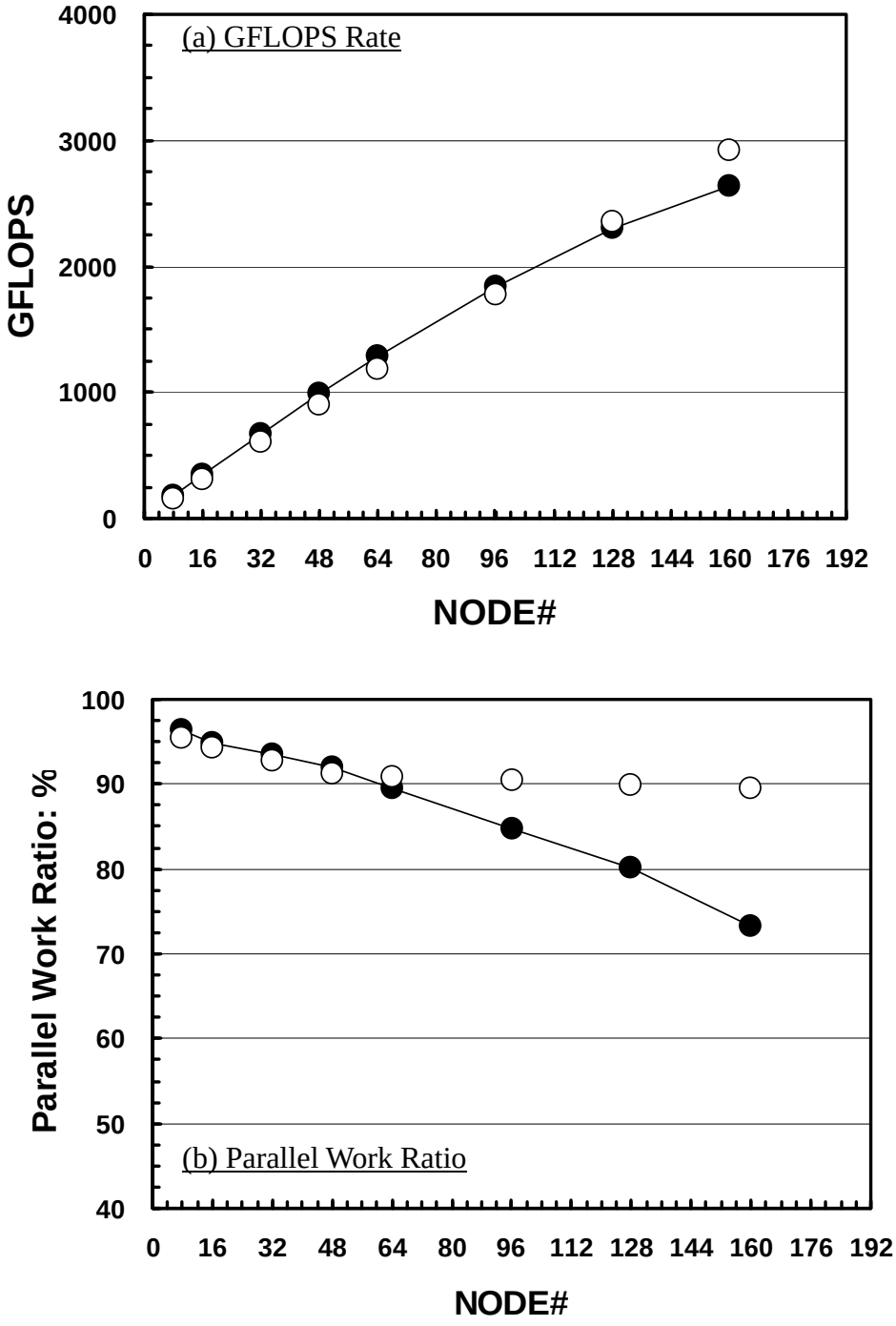


Fig. 3.31 Problem size and parallel performance on the Earth Simulator for the 3D linear elastic problem in Fig.3.10 using between 8 and 160 SMP nodes. (a)GFLOPS rate and (b)Parallel work ratio. Problem size/SMP node is fixed as 3,000,000 DOF (3×100^3). Largest case is 480,000,000 DOF on 160 SMP nodes (1280 PEs). Maximum performance is 2.63 (Flat MPI) and 2.92 (Hybrid) TFLOPS (Peak performance= 10.24 TFLOPS). (BLACK Circles: Flat MPI, WHITE Circles: Hybrid).PDJDS/CM-RCM reordering.

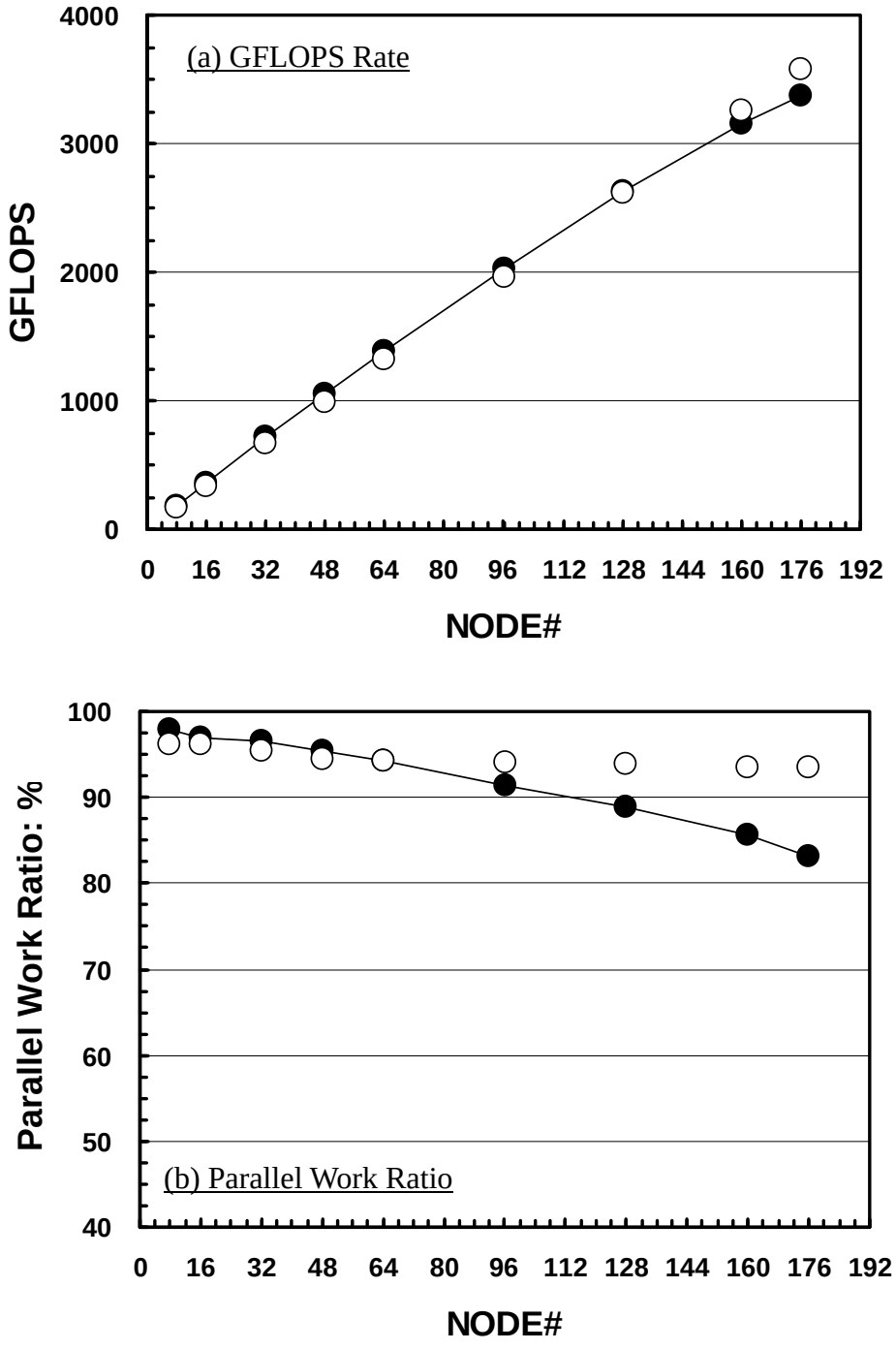


Fig. 3.32 Problem size and parallel performance on the Earth Simulator for the 3D linear elastic problem in Fig.3.10 using between 8 and 176 SMP nodes. (a)GFLOPS rate and (b)Parallel work ratio. Problem size/SMP node is fixed as 6,291,456 DOF (3×128^3). Largest case is 1,107,296,256 DOF on 176 SMP nodes (1408 PEs). Maximum performance is 3.35 (Flat MPI) and 3.58 (Hybrid) TFLOPS (Peak performance= 11.26 TFLOPS). (BLACK Circles: Flat MPI, WHITE Circles: Hybrid). PDJDS/CM-RCM reordering.

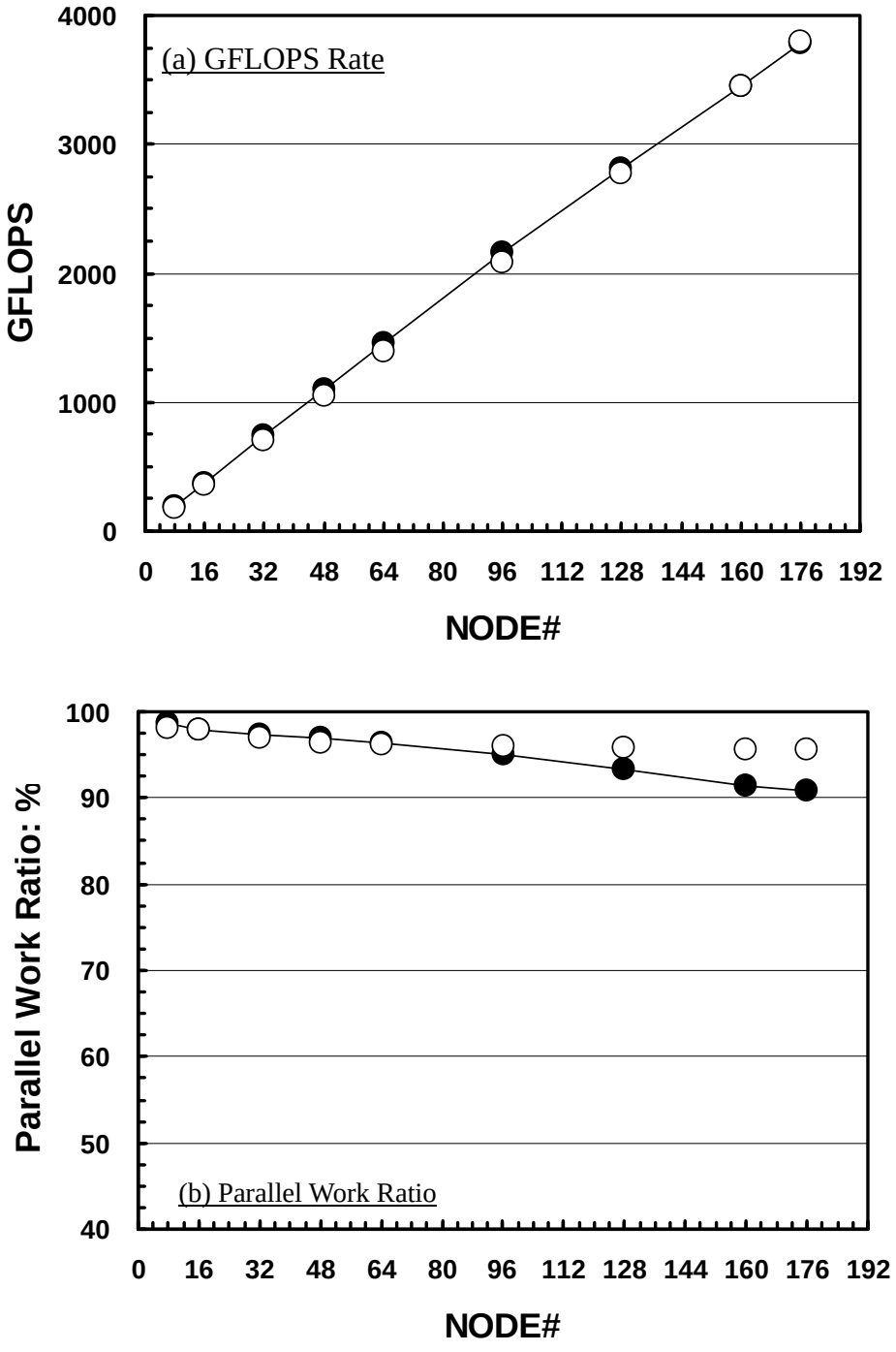
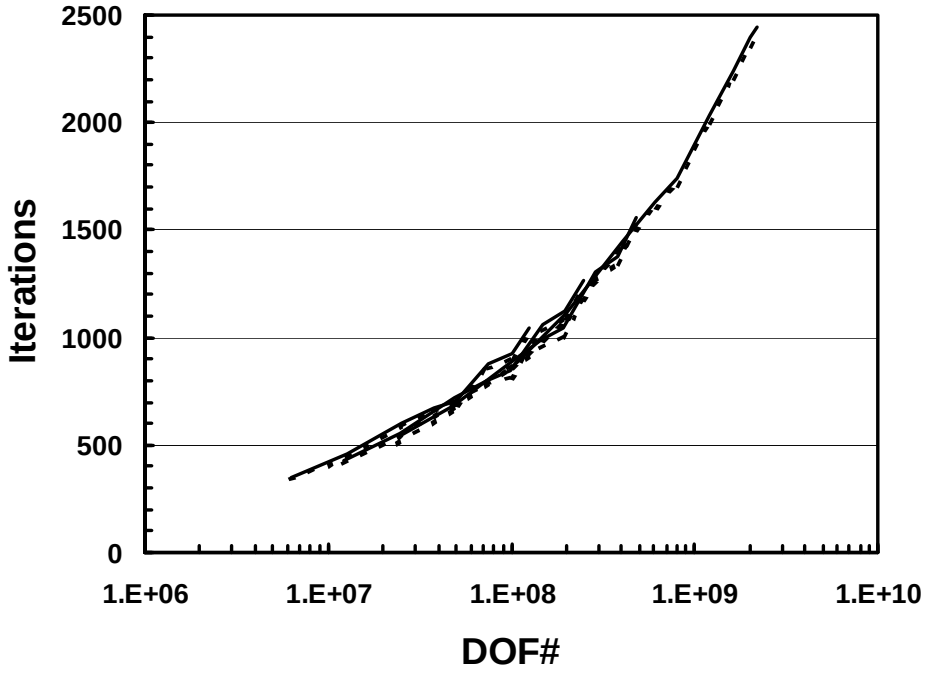


Fig. 3.33 Problem size and parallel performance on the Earth Simulator for the 3D linear elastic problem in Fig.3.10 using between 8 and 176 SMP nodes. (a)GFLOPS rate and (b)Parallel work ratio. Problem size/SMP node is fixed as 12,582,912 DOF (3×256×128×128). Largest case is 2,214,592,512 DOF on 176 SMP nodes (1408 PEs). Maximum performance is 3.78 (Flat MPI) and 3.80 (Hybrid) TFLOPS (Peak performance= 11.26 TFLOPS). (BLACK Circles: Flat MPI, WHITE Circles: Hybrid). PDJDS/CM-RCM reordering.

(a) Iterations for Convergence



(b) Hybrid/Flat MPI Performance Ratio

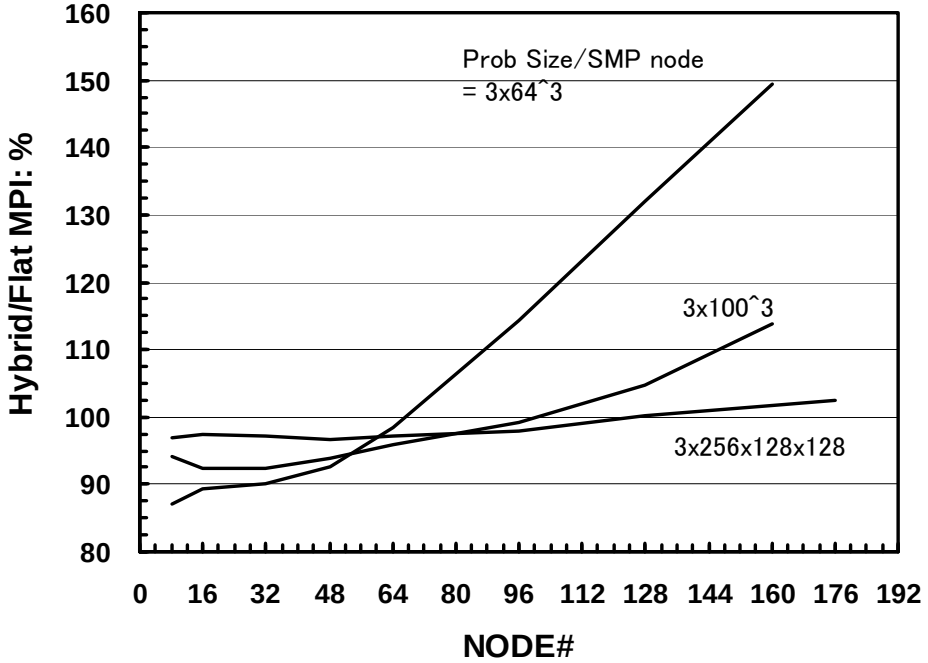
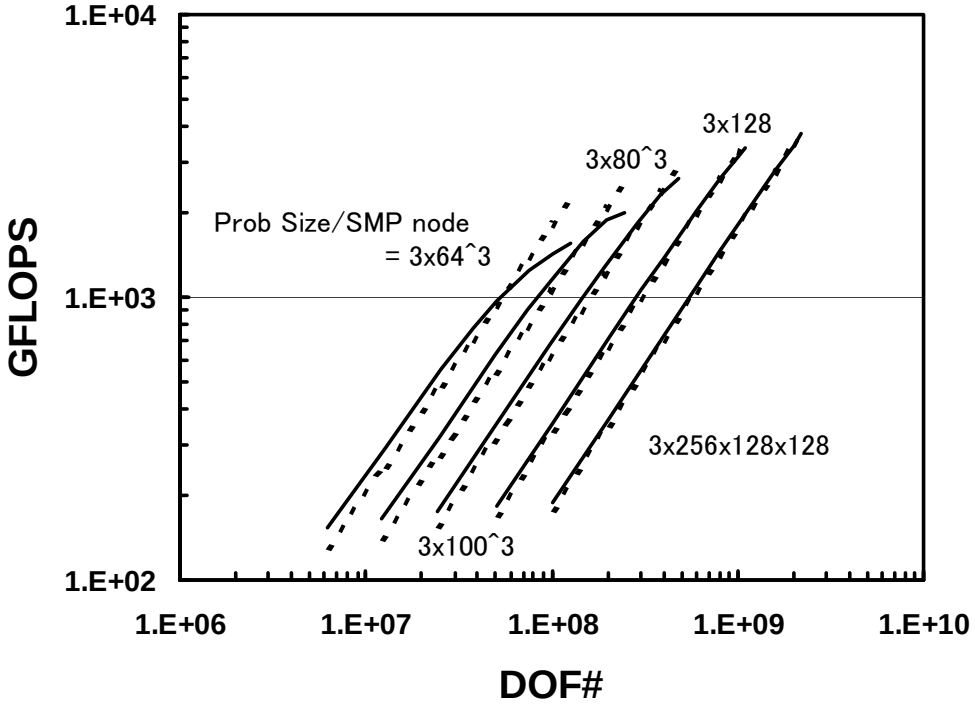


Fig. 3.34 Problem size and parallel performance on the Earth Simulator for the 3D linear elastic problem in Fig.3.10 using between 8 and 176 SMP nodes. (a) Iterations for convergence and (b) Hybrid/Flat MPI performance ratio based on elapsed time of Flat MPI. Problem size/SMP node is fixed (THICK lines: Flat MPI, DASHED lines: Hybrid in (a)). PDJDS/CM-RCM reordering.

(a) GFLOPS Rate



(b) Peak Performance Ratio

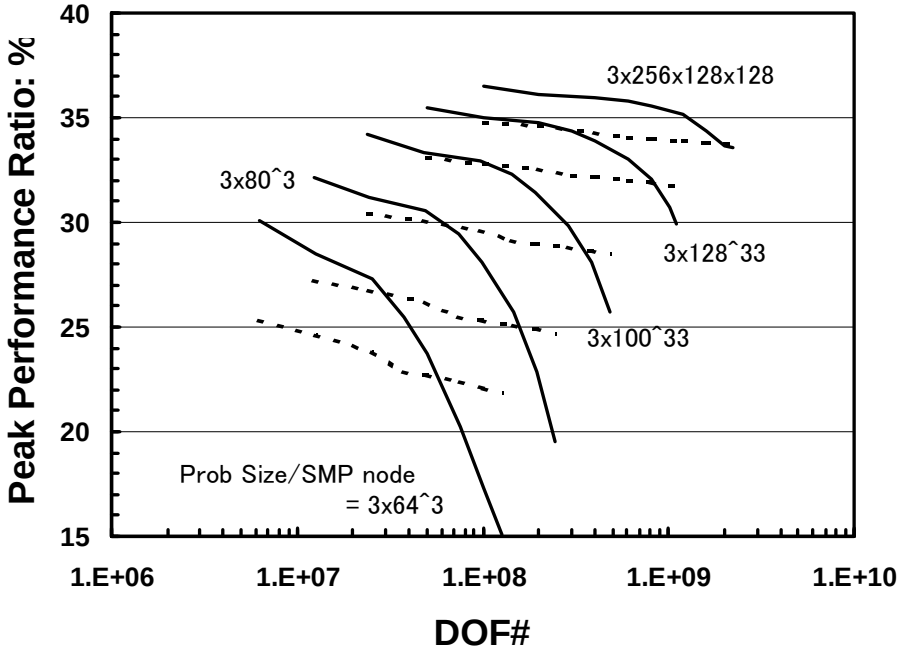


Fig. 3.35 Problem size and parallel performance on the Earth Simulator for the 3D linear elastic problem in Fig.3.10 using between 8 and 176 SMP nodes. (a) GFLOPS rate and (b) Ratio to the peak performance. Problem size/SMP node is fixed (THICK lines: Flat MPI, DASHED lines: Hybrid). PDJDS/CM-RCM reordering.

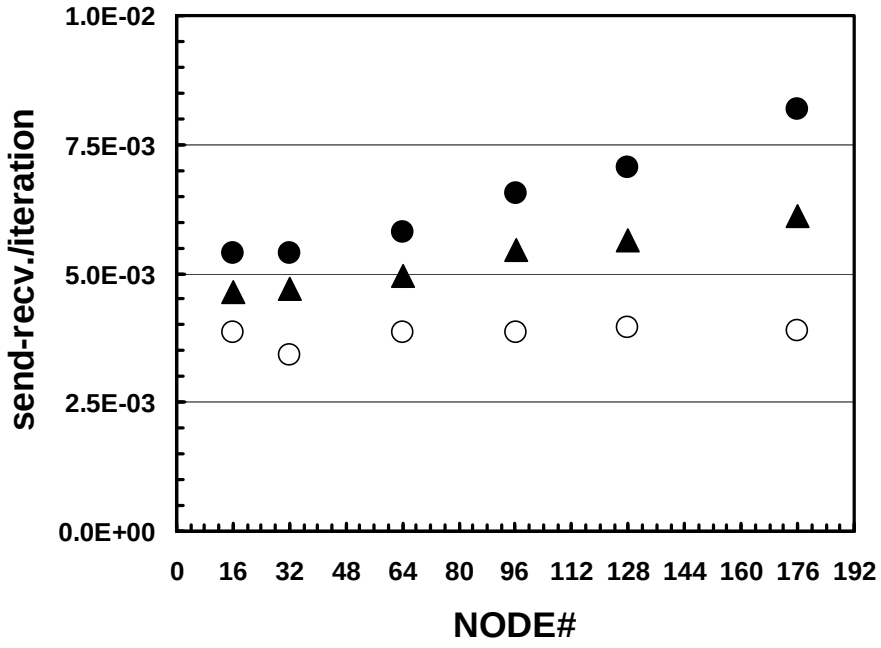


Fig. 3.36 Time (sec.) spent for communication subroutine per one iteration by the flat MPI programming model on the Earth Simulator for the 3D linear elastic problem in Fig.3.10 using between 8 and 176 SMP nodes. Problem size/SMP node is fixed as 6,291,456 DOF (3×128^3). (WHITE Circles: minimum, BLACK Circles: maximum, BLACK Triangles: average).

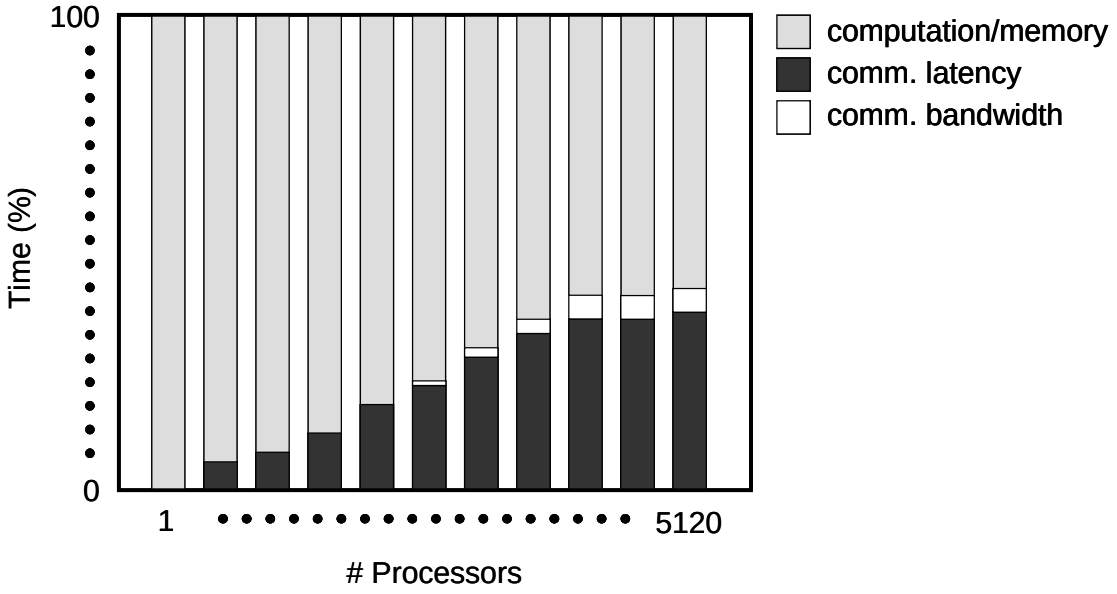


Fig. 3.37 Performance estimation of a finite-volume application code for CFD with local refinement on the Earth Simulator. Based on the results described in [45]. A greater percentage of time is taken by the latency component on larger processor counts, simply due to its much larger available bandwidth.

Chapter 4 Parallel Iterative Solvers with the Selective Blocking Preconditioning

In nonlinear problems such as contact simulations for geophysics, the condition numbers of the coefficient matrices are usually large due to special constraint conditions. The result is slow convergence of the iterative solver. In this study, a new preconditioning method, called *selective blocking* is developed along with a special partitioning method for parallel finite-element simulations on the GeoFEM platform. This newly developed method provides robust and smooth convergence and excellent parallel performance in 3D solid mechanics simulations for geophysics with contact conditions performed on a Hitachi SR2201 parallel computer with 128 processing elements using *flat MPI* parallel programming model. Finally, the method is vectorized and parallelized using OpenMP directives on one SMP node of the Earth Simulator.

4.1 Introduction

One of the most important applications of GeoFEM is simulation of ground motion. Stress accumulation on plate boundaries (faults) is very important in estimating the earthquake generation cycle (Fig.4.1).

In ground motion simulations, material, geometric and boundary nonlinearity should be considered. Boundary nonlinearity due to *fault-zone contact* is the most critical. In GeoFEM, the augmented Lagrange method (ALM) and penalty method are implemented, and a large penalty number λ is introduced for constraint conditions around faults [36]. The nonlinear process is solved iteratively by the Newton-Raphson (NR) method. Figure 4.2 shows the relationship between λ and the number of linear/nonlinear iterations [36]. In these cases, there is no friction on fault surfaces. Therefore, the coefficient matrices are symmetric positive definite [36], and GeoFEM's conjugate gradient iterative solver preconditioned by incomplete Cholesky factorization (ICCG) [21] was used for solving linear equations obtained at each Newton-Raphson cycle. This ICCG solver is a scalar version, in which all degrees of freedom (DOF) are treated independently. A large λ can provide an accurate solution and fast nonlinear convergence for the Newton-Raphson method, but the condition number of the coefficient matrices is large. Therefore, many iterations are required for convergence of the ICCG solver. Actually, the solver does not converge at all if $\lambda > 10^4$. Therefore, a more robust preconditioning method is essential for such ill-conditioned problems.

In this chapter, *selective blocking* is implemented to block type Krylov iterative solvers with ILU/IC preconditioning for fault-zone contact simulation. This method provides robust and efficient convergence. Moreover, a special partitioning method for parallel computation was developed in order to eliminate *edge-cuts* in contact groups. Parallel performance of this method is demonstrated on a Hitachi SR2201 parallel computer with up to 128 processing elements (PEs) using flat MPI parallel programming model. Developed procedure has been also optimized for the Earth Simulator using OpenMP and the results of this version are also presented. In this chapter, we will provide a brief overview of the *selective blocking* along with special partitioning, and show some examples.

4.2 Preconditioning Methods for Ill-Conditioned Problems

4.2.1 Preliminary Results

Linear equations derived from actual nonlinear contact problems were solved using the CG method of GeoFEM with various types of preconditioners using a single processor (COMPAQ Alpha 21164-600MHz). These equations were obtained at a certain Newton-Raphson cycle in nonlinear contact problems for the finite element model, as shown in Fig.4.1. λ is the normalized penalty number, say the penalty number divided by Young's modulus (E). Coefficient matrices are symmetric positive definite for 3D elastic contact problems if no friction exists on fault surfaces.

The first two items of Table 4.1 show the results obtained using GeoFEM's original scalar CG solver preconditioned by diagonal scaling and IC with no fill-in. In GeoFEM's original scalar solver, all DOF are treated independently. In these cases, the iterative solver converges fast if $\lambda=10^0$, but does not converge at all if $\lambda=10^6$.

The typical remedies using an ILU/IC type of preconditioning method for ill-conditioned matrices are as follows:

- Blocking
- Deep Fill-in
- Reordering.

4.2.2 Blocking

First of all, 3×3 block operation was introduced for 3D solid mechanics. In 3D solid mechanics problems, three DOF exist on each finite-element node. Full LU factorization is introduced in this 3×3 block. Thus, by using a block ILU/IC preconditioning (BILU/BIC), three DOF on the same finite-element node can be treated in a more simultaneous manner than by using the original *scalar* ILU/IC preconditioning in GeoFEM. Figure 4.3 shows how the forward substitution procedure of BILU/BIC, with no fill-in, can be written in FORTRAN.

4.2.3 Deep Fill-in

Another remedy for ill-conditioned matrices – *deep fill-in* – applies a level of fill-in to block ILU/IC preconditioning (BILU(*n*)/BIC(*n*), where *n* is the level of fill-in). Procedure of LU factorization or Gaussian elimination is as follows [7,21,103] :

Gaussian Elimination

```
do i= 2, n
  do k= 1,  i-1
    a_jk := a_jk/a_kk
    do j= k+1, n
      a_ij := a_ij - a_ik*a_kj
    enddo
  enddo
enddo
```

In this procedure, many fill-in occurs during factorization, therefore factorized matrix could be dense even if original matrix is sparse [103]. ILU(n) or IC(n) are incomplete factorization where *n*-level fill-in is allowed. Larger *n* provides more accurate factorization and usually leads to robust preconditioning, but more expensive in both memory and CPU time. In many engineering applications, ILU(0)/IC(0) is widely used where there are no fill-in and non-zero pattern of factorized matrix is kept as original coefficient matrix:

ILU(0)

```
do i= 2, n
  do k= 1,  i-1
    if ((i,k) ∈ NonZero(A)) then
      a_jk := a_jk/a_kk
    endif
    do j= k+1, n
      if ((i,j) ∈ NonZero(A)) then
        a_ij := a_ij - a_ik*a_kj
      endif
    enddo
  enddo
enddo
```

ILU(n)/IC(n) with n-level fill-in is described as follows:

ILU(n)

LEV_{ij}=0 if ((i,j) $\in$ NonZero(A)) otherwise LEV_{ij}= p+1
do i= 2, n
  do k= 1, i-1
    if ($LEV_{ik} \leq p$) then
      $a_{jk} := a_{jk}/a_{kk}$
    endif
    do j= k+1, n
      if ($LEV_{ij} = \min(LEV_{ij}, 1+LEV_{ik}+ LEV_{kj}) \leq p$) then
        $a_{ij} := a_{ij} - a_{ik}*a_{kj}$
      endif
    enddo
  enddo
enddo

4.2.4 Selective Blocking

In addition to deep fill-in, a special method called *selective blocking* was also developed for contact problems [75,77]. In the *selective blocking* method, strongly coupled finite-element nodes in the same contact group [36] coupled through penalty constraints are placed into the same large block (*selective block* or *super* node) and all of the nodes involved are reordered according to this blocking information. Full LU factorization is applied to each selective block where the size of the block is (3×NB) × (3×NB) in 3D problems and NB is the number of finite-element nodes in the selective block, as shown in Fig.4.4 and Fig.4.5. Thus, local equations for coupled finite-element nodes in contact groups are solved by means of a *direct* method during preconditioning.

This idea (selective blocking) is also related to the *clustered element-by-element method* (CEBE) described in [18,50] or *Blocked ICT* preconditioning in [82], where the elements are partitioned into clusters of elements, with a desired number of elements in each cluster, and the iterations are performed in a cluster-by-cluster fashion. The method is highly suitable for both vectorization and parallelization if used with proper clustering and element grouping schemes. Any number of elements can be brought together to form a cluster, and the number of elements should be viewed as an optimization parameter to minimize computational cost. The CEBE method becomes equivalent to the direct method when the cluster size is equal to the total number of elements. Generally, larger clusters provide better convergence rates because a larger number of fill-in elements are taken into account during factorization, but the cost per iteration cycle increases according to the size of the cluster, as shown in Fig.4.6. The trade-off between convergence and computational cost is not clear, but the results of examples in [18,50,82] show that larger clusters provide better performance.

In selective blocking, clusters are formed according to contact group information. Usually, the size of each cluster is much smaller than that in a general CEBE method. If a finite element node does not belong to any contact group, then the finite element node forms a cluster of which the size is equal to that in selective blocking. Therefore, computational cost for *selective blocking* during one iteration cycle is equivalent to that of BILU(0)/BIC(0) if fill-in between clusters is not considered.

4.2.5 Evaluation of Developed Methods

By introducing the 3×3 block, the CG solver preconditioned by block IC with no fill-in (BIC(0)) converges even when λ is as large as 10^6 (Table 4.1). *Deep fill-in* options provide faster convergence but the SB-BIC(0) (BIC(0) preconditioning with the *selective blocking* reordering) shows the best performance (Table 4.1).

SB-BIC(0) usually requires a greater number of iterations for convergence compared to BIC(1) and BIC(2) but the overall performance is better because the computation time for each iteration is much shorter. Because no *inter-block* fill-in is considered in SB-BIC(0), the memory requirement in this method is usually as small as that in BIC(0) with no fill-in. Only the *inter-node* fill-in in each *selective block* is considered in SB-BIC(0).

Krylov iterative solver with *selective blocking* preconditioning can be considered to be a hybrid of iterative and direct methods where local equations for coupled finite-element nodes in contact groups are solved by means of a direct method during preconditioning. This method combines the efficiency and the scalability of iterative methods with the robustness of direct methods.

4.3 Strategy for Parallel Performance

Localized ILU/IC [72,73] is an efficient parallel preconditioning method, but it is not robust for ill-conditioned problems. Table 4.2 shows the results by parallel CG solvers with localized preconditioning on a 4 PE workstation cluster using distributed matrices created by k-way METIS [138], for the problem described in Fig.4.1. According to the results, the number of iterations for convergence increases by a factor of 10 in $\lambda=10^6$ cases. This is because the *edge-cuts* occur at inter-domain boundary edges that are included in contact groups [72,73].

In order to eliminate these edge-cuts, a partitioning technique has been developed so that all nodes which belong to the same contact group are in the same domain. Moreover, nodes are re-distributed so that load-balancing among domains should be attained for efficient parallel computing (Fig. 4.7).

In GeoFEM, there are several types of special elements for contact problems (types 411, 412, 421, 422, 511, 512, 521 and 522) [131]. Nodes included in the same elements of these types are connected through penalty constraints and form a contact group. In the new partitioning method, the partitioning process is executed so that these nodes in the same contact elements are on the same domain, or PE. These functions are added to the original domain partitioner in GeoFEM.

Table 4.3 shows the results obtained by this partitioning method. The number of iterations for convergence has been dramatically reduced for each preconditioning method although it is larger than that of the single PE cases, as shown in Table 4.1 due to localization.

4.4 Benchmarks

4.4.1 Overview

The efficiency and robustness of the developed preconditioning and partitioning methods for simulations of fault-zone contact were evaluated in two types of 3D applications.

Figure 4.8 shows the simple geometry and boundary conditions of an example model for 3D linear elastic solid mechanics. In this example, linear multiple point constraint (MPC) conditions were applied to the nodes of contact groups in the following manner:

- The locations of nodes in each contact group are identical.
- All nodes in the contact groups are coupled tightly in any direction on the surfaces.
- Infinitesimal linear elastic deformation theory in solid mechanics was applied. Therefore nodes do not move, and the contact relationships have been kept during the simulation.
- A penalty constraint is applied to the nodes in the contact groups. 111-type element (Rod/Beam) in GeoFEM [131] is put in each contact group (Fig.4.9) and very large stiffness corresponding to penalty is applied. Figure 4.10 shows the matrix operation of nodes in a contact group. Three components in x, y, and z directions are constrained through penalty.

A large penalty parameter provides a stronger constraint but the condition numbers of the coefficient matrices are larger. Therefore, the convergence of iterative solvers is usually slow if the penalty is large. The problem itself is linear elastic, but solving linear equations by iterative methods is as difficult as solving equations in nonlinear contact problems, such as those shown in previous sections. The definitions of the model and boundary conditions are as follows:

- Three zones of uniform material property for which non-dimensional E (Young's modulus)=1.0, ν (Poisson ratio)=0.30. Tri-linear (1st order) cubic hexahedral elements are used for spatial discritization.
- Uniform MPC conditions were imposed on the nodes along the boundary

surfaces of the blocks. Therefore, the number of nodes in each contact group can be different, as shown in Fig.4.8(b).

- Symmetry boundary conditions were applied at the x=0 and y=0 surfaces.
- Free boundary conditions were applied at the $x=X_{max}$ and $y=Y_{max}$ surfaces.
- Dirichlet (fixed) boundary conditions were applied at the z=0 surface.
- A uniformly distributed load in the z-direction was applied at the $z=Z_{max}$ surface.
- If friction is not considered at fault surfaces, the coefficient matrix is symmetric positive definite; therefore, the CG method was adopted.

All of the meshes in this example are uniform cubes.

The second example, as shown in Fig.4.11, is a more complicated model for earthquake simulation in the southwestern part of Japan [36]. This model consists of crust (dark gray) and subduction plates (light gray). 27,195 nodes and 23,831 tri-linear (1st-order) hexahedral elements are included. The same boundary conditions as those used in the model of Fig.4.8 were applied here. In this Southwest Japan model, a body force of -1.0 was applied in the z-direction rather than a surface force at the $z=Z_{max}$ surface, as in Fig.4.8, and no symmetry boundary conditions were applied in the x or y directions. In this example, meshes are irregular, and some of the meshes are very distorted. The material property is linear and homogeneous (E (Young's modulus)=1.0, ν (Poisson ratio)=0.30).

In the following part of this section, the results of benchmarks using two types of models on a single processor (COMPAQ Alpha 21164-600 MHz) are shown, and then the parallel performance of the method is described by large-scale computation on a Hitachi SR2201 using *flat MPI* parallel programming model.

4.4.2 Benchmarks-1 (Simple Block Model)

(1) Benchmarks

First, benchmarks for the simple block model, as shown in Fig.4.8, have been conducted for a wide range of penalty parameter values using various types of preconditioners on a single processor (COMPAQ Alpha 21164-600MHz). In the benchmarks, the following model is considered:

- NX1=20, NX2=20, NY =15, NZ1= 20, NZ2= 20 (Fig.4.8(a))
- Total Elements = 24,000, Total Nodes = 27,888, Total DOF = 83,664.

Table 4.4 shows the results for convergence. BIC(0) does not converge if λ is larger than 10^4. BIC(1), BIC(2) and SB-BIC(0) provide robust convergence for a wide range of λ values. SB-BIC(0) provides the most efficient performance although the iteration number for convergence is larger than that for BIC(1) and BIC(2).

(2) Eigenvalue analysis

Next, the robustness of the preconditioning method was estimated according to the eigenvalue distribution of the $[M]^{-1}[A]$ matrix by the method in [7,21], where [A] is the original coefficient matrix and $[M]^{-1}$ is the inverse of the preconditioning matrix.

In a symmetric positive definite matrix, the spectral condition number κ is given by $\kappa=E_{max}/E_{min}$ where E_{max} and E_{min} are the largest and smallest eigenvalues, respectively, of $[M]^{-1}[A]$ [7,21].

Table 4.5 shows E_{min}, E_{max} and κ derived from each preconditioning method for a range of penalty parameter values. According to Table 4.5, all of the eigenvalues are approximately constant and close to 1.00 for a wide range of λ values except for BIC(0). BIC(1) and BIC(2) provide a slightly smaller κ than SB-BIC(0).

4.4.3 Benchmarks-2 (Southwest Japan Model)

Benchmarks of the Southwest Japan model with complicated geometry, as shown in Fig.4.11, have been conducted for a wide range of penalty parameter values using various types of preconditioners on a single processor (COMPAQ Alpha 21164-600 MHz).

Table 4.6 shows the results for convergence. BIC(0) does not converge if λ is larger than 10^4. BIC(1), BIC(2) and SB-BIC(0) provide robust convergence for a wide range of λ values but the iteration number for convergence increases in BIC(1) and BIC(2) as λ changes from 10^2 to 10^4.(BIC(1) from 201 to 259, BIC(2) from 176 to 232). SB-BIC(0) provides the most efficient performance although the iteration number for convergence is larger than both BIC(1) and BIC(2).

Table 4.7 shows E_{min}, E_{max} and κ derived from the eigenvalue analysis of $[M]^{-1}[A]$ for each preconditioning method. In SB-BIC(0), E_{min}, E_{max} and κ remain constant for a wide range of λ values but κ increases in BIC(1) and BIC(2) when λ changes from 10^2 to 10^4. This corresponds to the increase in the iteration number for convergence in BIC(1) and BIC(2) between $\lambda=10^2$ and $\lambda=10^4$. In this example, the geometry is much more complicated than in the previous simple block model. Moreover, meshes are not uniform and some of the meshes are distorted. The distortion of an individual mesh directly affects the components of the coefficient matrix [A] for linear equations and the

eigenvalue distribution of $[M]^{-1}[A]$. SB-BIC(0) is robust under such conditions.

In both models (simple block and Southwest Japan), the spectral condition number of $[M]^{-1}[A]$ is a helpful parameter for the evaluation of the convergence of the preconditioning methods. In the simple block model, the spectral condition number of $[M]^{-1}[A]$ by BIC(1) and BIC(2) is usually smaller than that of SB-BIC(0) and the iteration number for convergence is smaller (Tables 4.4 and 4.5). In contrast, the Southwest Japan model provides a larger spectral condition number for BIC(1) and BIC(2) than in SB-BIC(0) when λ is larger than 10^4, but the iteration number for the convergence of BIC(1) and BIC(2) is smaller than that for SB-BIC(0) (Tables 4.6 and 4.7).

4.4.4 Large-Scale Computation by Flat MPI

(1) Simple Block Model

A large-scale computation was performed on the simple block model, as shown in Fig.4.8. The following specific values describe the model:

- NX1=70, NX2=70, NY =40, NZ1= 70, NZ2= 70 (Fig.4.8(a))
- Total Elements= 784,000, Total Nodes= 823,813, Total DOF= 2,471,439.

This example problem was solved by parallel iterative solvers using various types of preconditioning methods for various penalty numbers for the MPC conditions. Domains are partitioned according to the contact group information described in the previous chapter. Computations were performed using 16 to 128 PEs on a Hitachi SR2201 at the University of Tokyo using flat MPI parallel programming model described in Chapter 1.

Table 4.8 shows the results for various preconditionings. BIC(0) does not converge if λ is larger than 10^4. BIC(1), BIC(2) and SB-BIC(0) provide robust convergence for a wide range of λ values. SB-BIC(0) provides the most efficient performance, although the iteration number for convergence is larger than BIC(1) and BIC(2). Table 4.9 and Fig.4.12 show the parallel performance for the same problem solved using 16 to 128 PEs of Hitachi SR2201. BIC(1) did not work if the PE number was less than 64 and BIC(2) worked only for 128 PEs due to memory limitation. As shown in Table 4.9 and Fig.4.12, the iteration number for convergence increases according to PE number in BIC(0) and SB-BIC(0) due to the locality of the preconditioning method, but this increase is very slight (only 11% increase from 16 PEs to 128 PEs). The speed-up ratio based on elapsed execution time including communication for 128 PEs, is more than 120, as extrapolated from the results obtained

using 16 PEs.

The required memory size for each preconditioning method is compared in Fig.4.13 for this problem. The memory size of each PE on the Hitachi SR2201 is 256 MB but only 224 MB of the memory is available for users. For example, BIC(2) does not function on 64 PEs because the required memory size is 14.4 GB but only 14.3 GB (224×64/1000=14.34) are available on 64 PEs. The required memory size for SB-BIC(0) is competitive with that of BIC(0), is less than 50% of that of BIC(1), and approximately 25% of BIC(2). The required memory size for SB-BIC(0) could change, according to the number of contact groups and the size of the matrix blocks by selective blocking, but the required memory size is much less than that of BIC(1) or BIC(2) because block-to-block fill-in is not considered in SB-BIC(0).

(2) Southwest Japan Model

Finally, examples with a more complicated Southwest Japan model were solved by the parallel iterative solvers preconditioned by SB-BIC(0) on a Hitachi SR2201 using 16 to 128 PEs. The model, as shown in Fig.4.11, has been globally refined and the final mesh obtained has 997,422 nodes and 960,509 elements. The total number of DOF is 2,992,264.

Table 4.10 and Fig.4.14 show the parallel performance for the same problem solved using 16 to 128 PEs of Hitachi SR2201. The iteration number for convergence increases according to the number of PE due to the locality of the preconditioning method, but this increase is very slight (only 15% increase from 16 PEs to 128 PEs). The speed-up ratio based on elapsed execution time including communication for 128 PEs is more than 107, as extrapolated from the results obtained using 16 PEs.

4.5. Optimization for the Earth Simulator

4.5.1 Overview

In this section, *selective blocking* is ported to SMP cluster architectures with vector processors such as the Earth Simulator. Hybrid parallel programming model is adopted with reordering methods for vector and parallel performance [72,73]. Parallel and vector performance of this method is demonstrated on the Earth Simulator with a single SMP node.

4.5.2 Reordering Methods for Parallel/Vector Performance on SMP Nodes

In order to achieve efficient parallel/vector computation for applications with unstructured grids, the following 3 issues are critical:

- Local operations and no global dependency
- Continuous memory access
- Sufficiently long loops

For unstructured grids, in which data and memory access patterns are very irregular, reordering technique is very effective for achieving highly parallel and vector performance, as describe in Chapter 3. The same strategy as that in Chapter 3 is adopted here.

According to the results in Chapter 3, PDJDS/CM-RCM reordering, which is a method combining cyclic multicoloring and RCM (Reverse Cuthil-Mckee) reordering, provides fast and robust convergence for simple geometries; however, for complicated geometries in real-world applications, the number of hyperplanes may be extremely large [79,81], and constructing independent sets having a sufficiently large loop length by cyclic multicoloring (CM) is usually very difficult. Under these circumstances, classical multicoloring (MC) offers another option. Although MC usually provides slower convergence than CM-RCM and RCM, a sufficiently large loop length is guaranteed when a certain number of colors is specified. In this work, the PDJDS/MC reordering method was adapted in order to achieve higher vector performance.

4.5.3 Special Treatments for Selective Blocking

In selective blocking preconditioning, individual selective blocks (or super nodes) are computed independently, therefore dependency among selective blocks should be considered at reordering for vector optimization. In this case, load imbalance may occur because the size of each selective block differs according to the number of nodes in each contact group. Ordinary nodes which do not belong to any contact group are considered as a selective block of size one. Currently, no special treatment for load-balancing is implemented.

Another problem is that the number of off-diagonal components may not reduce smoothly in DJDS reordering according to the size of the contact groups and the number of connected nodes, as shown in Fig.4.15. In this case, dummy elements are placed so as to maintain a smooth decrease in the number of off-diagonal components in descending order. If several dummy elements exist, efficiency and load balancing may be affected.

Finally, block diagonal components for selective blocks are reordered according to block size on each PE and for each color, as shown in Fig.4.16. Thus, *if* statements according to block size are eliminated from the full LU factorization procedure for each selective block during back/forward substitution.

4.5.4 Results

Figure 4.17 and 4.18 show the results for the simple block model and Southwest Japan model, respectively. In the cases with many colors, fewer iterations are required for convergence, but the performance is worse due to the smaller loop length and greater overhead. In the Southwest Japan model, iterations for convergence is not affected by number of colors. This is because there are many distorted elements in this model and the coefficient matrices are ill-conditioned. Performance of 17.6 GFLOPS (27.5% of peak performance, 64 GFLOPS) for the simple block model and 18.6 GFLOPS (29.1% of peak performance) for the Southwest Japan has been obtained.

Figure 4.19 compares the performance with results obtained by the method without the reordering selective blocks according to block size, as shown in Fig.4.16. Performance is about 60% if this reordering is not applied. Figure 4.20 shows load-imbalance among PEs on the SMP node and ratio of dummy off-diagonal components. Load-imbalance is computed by the following method:

$$\text{Load Imbalance (\%)} = 100 \times (\text{max. node \#} - \text{min. node \#}) / \text{average node \#}$$

Effect of load-imbalance and dummy elements are very small in both models and effect is negligible in this computation.

4.6. Summary

In this chapter, robust preconditioning and partitioning methods were developed for the simulation of fault-zone contact with penalty constraints using parallel iterative solvers. For symmetric positive definite matrices, block incomplete Cholesky factorization without inter-block fill-in, using *selective blocking* (SB-BIC(0)) has excellent performance, memory efficiency and robustness for a wide range of penalty parameter values even if meshes are distorted. Spectral condition number κ ($\kappa=E_{max}/E_{min}$ where E_{max} and E_{min} are the largest and smallest eigenvalues, respectively, of $[M]^{-1}[A]$) is a helpful parameter for the evaluation of convergence of the preconditioning methods. Usually, BIC(1) and BIC(2) requires fewer iterations for convergence than SB-BIC(0). However, the total computation time for SB-BIC(0) is lower as a result of the lower cost per iteration.

It is also shown that the partitioning method for elimination of edge-cuts in contact groups with load-balancing improves the convergence of parallel iterative solvers with localized preconditioning.

Parallel performance of the CG method with SB-BIC(0) preconditioning was evaluated using 16 to 128 PEs of a Hitachi SR2201 at the University of Tokyo using a flat MPI parallel programming model. Although the iteration number for convergence increases according to PE number due to locality of the preconditioner, this increase is only 11% from 16 PEs to 128 PEs and the speed-up ratio based on elapsed execution time including communication for 128 PEs, is higher than 120, as extrapolated from results for 16 PEs.

Furthermore, the developed method is vectorized and parallelized using OpenMP directives on one SMP node of the Earth simulator, and provides robust and smooth convergence and excellent parallel performance for both simple and complicated geometries with contact conditions.

The reordering method for SMP cluster architectures with vector processors described in Chapter 3 has been implemented to the selective blocking preconditioning using the MC reordering method. Special treatments for selective blocking, such as the introduction of dummy elements and the reordering of selective blocks according to block size, were implemented.

In cases involving several colors, fewer iterations are required for convergence, but the performance is worse due to the smaller loop length and greater overhead. In the complicated Southwest Japan model, the number of iterations for convergence is not

affected by the number of colors because there are many distorted elements in this model and the coefficient matrices are ill-conditioned.

Performance of 17.6 GFLOPS (27.5% of peak performance) for the simple block model and 18.6 GFLOPS (29.1% of peak performance) for the Southwest Japan has been obtained. Performance is about 60% if the reordering of selective blocks is not applied. The load-imbalance among PEs on the SMP node and the ratio of dummy off-diagonal components are not significant.

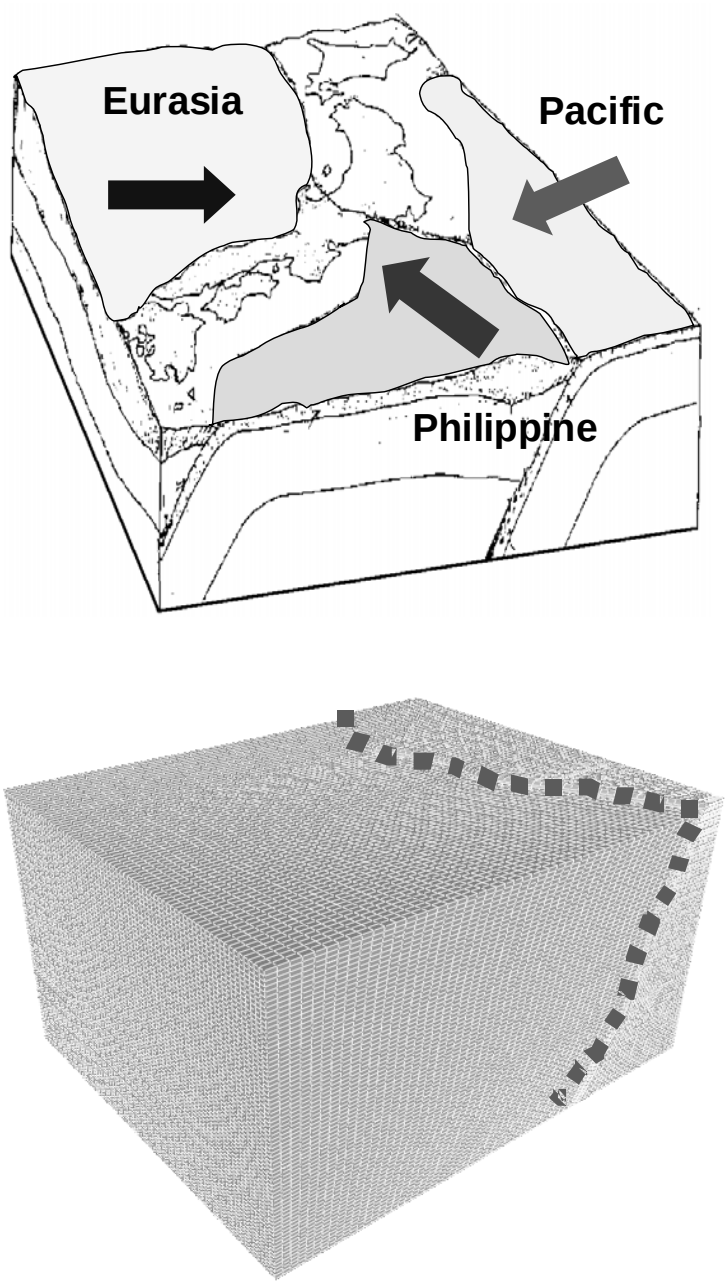


Fig. 4.1 Plate boundaries (faults) around Japanese Islands and an example of the finite element model (6,156 tri-linear (1st order) hexahedral elements, 7,220 nodes, 21,660 DOF, 840km×1020km×600km region) [36,131]

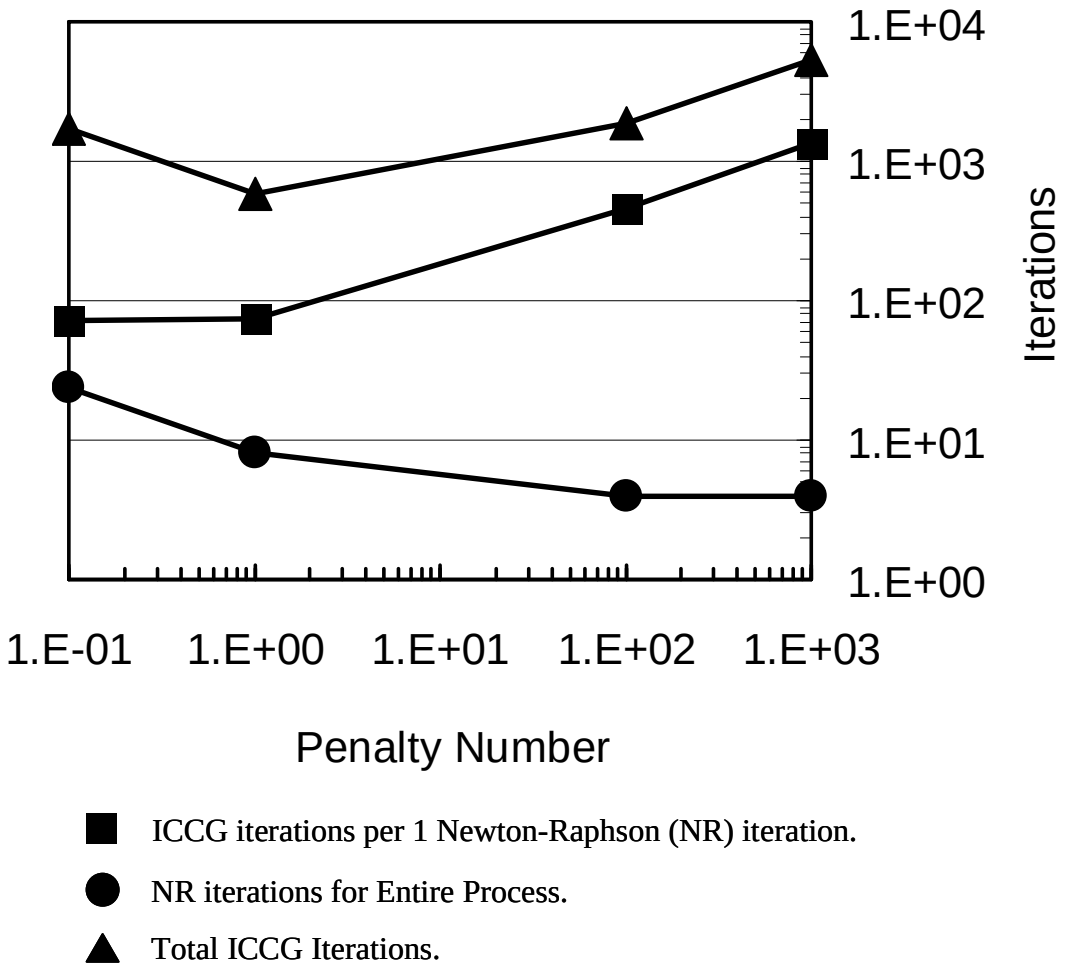


Fig. 4.2 Typical λ (penalty number)-iterations relationship in fault-zone contact computation without friction by ALM [36] (circle: Newton-Raphson (NR) iterations, square: ICCG iterations per one NR iteration, triangle: Total ICCG iterations)

Table 4.1 Iterations/CPU time (includes factorization) for convergence ($\varepsilon=10^{-8}$) on a single PE of COMPAQ Alpha 21164/600MHz by preconditioned CG for the 3D elastic fault-zone contact problem in Fig.4.1 (21,660 DOF).: **BIC(n)**: Block IC with n-level fill-in, **SB-BIC(0)**: BIC(0) with the selective blocking reordering.

Preconditioning	λ	Iter #	sec.
Diagonal	10^0	340	19.1
Scaling	10^6	No Conv.	N/A
IC(0)	10^0	85	8.9
(Scalar Type)	10^6	No Conv.	N/A
BIC(0)	10^0	82	8.8
	10^6	1,108	116.8
BIC(1)	10^0	44	9.4
	10^6	94	17.9
BIC(2)	10^0	32	12.2
	10^6	33	13.0
SB-BIC(0)	10^0	78	9.4
	10^6	77	9.4

$Mp=q$ where $M=(L+D)D^{-1}(D+U)$

Forward Substitution

$(L+D)p= q : p= D^{-1}(q-Lp)$

Backward Substitution

$(I+ D^{-1} U)p_{new}= p_{old} : p= p - D^{-1} Up$

```
do i= 1, N
  SW1= Z(3*i-2)
  SW2= Z(3*i-1)
  SW3= Z(3*i  )
  isL= INL(i-1)+1
  ieL= INL(i)
  do j= isL, ieL
      k= IAL(j)
     X1= Z(3*k-2,Z)
     X2= Z3*k-1,Z)
     X3= WW(3*k  ,Z)
    SW1= SW1 - AL(1,1,j)*X1 - AL(1,2,j)*X2 - AL(1,3,j)*X3
    SW2= SW2 - AL(2,1,j)*X1 - AL(2,2,j)*X2 - AL(2,3,j)*X3
    SW3= SW3 - AL(3,1,j)*X1 - AL(3,2,j)*X2 - AL(3,3,j)*X3
  enddo
```

Full LU factorization for 3x3 block

```
  X1= SW1
  X2= SW2
  X3= SW3
  X2= X2 - ALU(2,1,i)*X1
  X3= X3 - ALU(3,1,i)*X1 - ALU(3,2,i)*X2
  X3= ALU(3,3,i)*  X3
  X2= ALU(2,2,i)*( X2 - ALU(2,3,i)*X3 )
  X1= ALU(1,1,i)*( X1 - ALU(1,3,i)*X3 - ALU(1,2,i)*X2)
  WW(3*i-2,Z)= X1
  WW(3*i-1,Z)= X2
  WW(3*i  ,Z)= X3
enddo
```

```
WW (:,Z) : work vector
INL(:)   : coefficient index of the lower triangular matrix (LTM)
IAL(:)   : connected nodes of LTM

AL (3,3,:): 3x3 coefficient matrix component of the LTM
ALU(3,3,:): 3x3 LU factorization for each 'node'
```

Fig. 4.3 Procedure of the 3×3 block ILU(0) preconditioning: forward substitution [75,77]

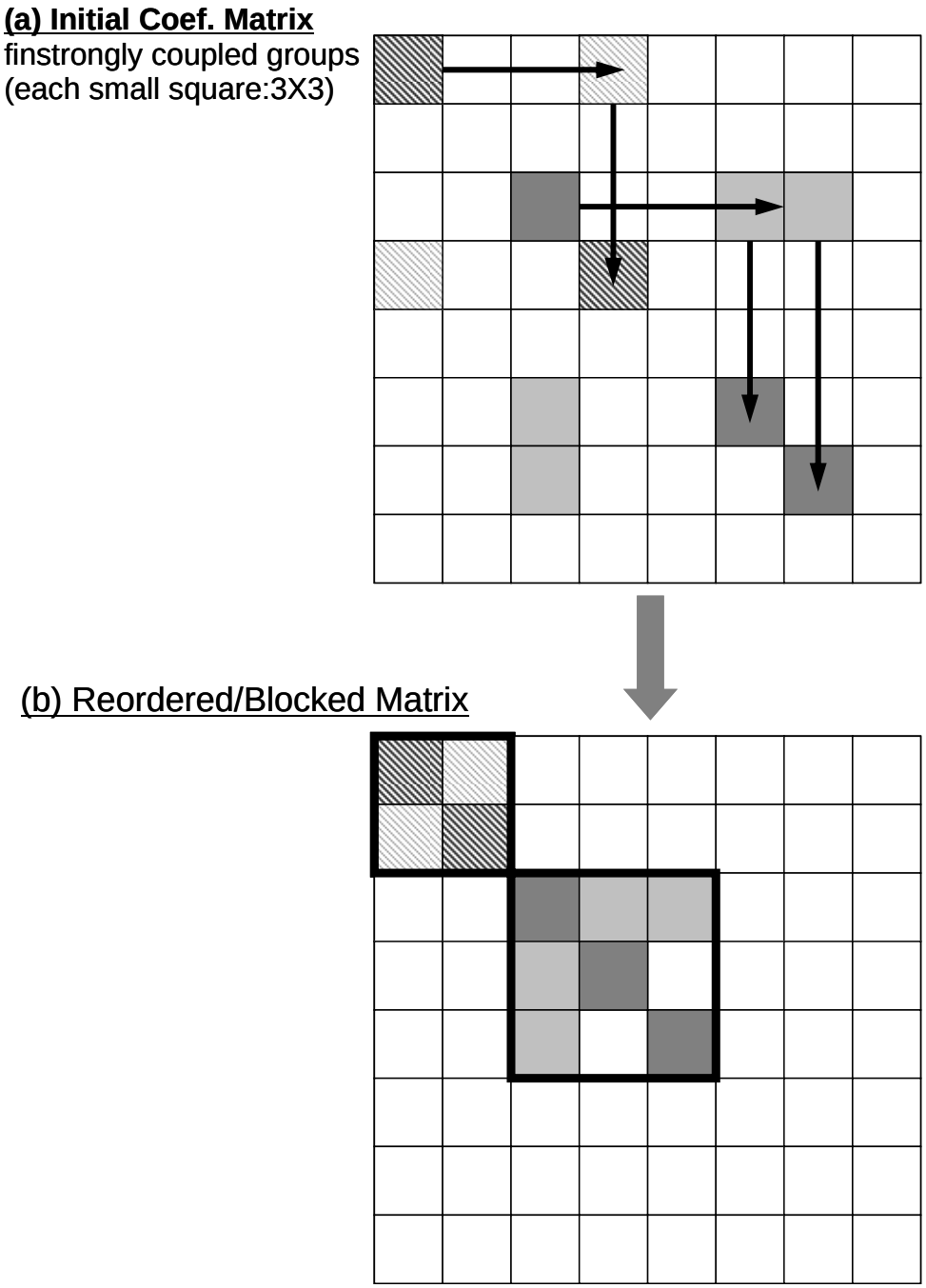


Fig. 4.4 Procedure of the *selective blocking* : Strongly coupled elements are put into the same *selective block*. (a) searching for strongly coupled components and (b) reordering and selective blocking. Full LU factorization procedure is applied to each *selective block*. Coupled finite-element nodes in contact groups can be solved in direct method during preconditioning procedure. In SB-BIC(0) (BIC(0) preconditioning combined with the *selective blocking* reordering), no *inter-block* fill-in is considered. Only *inter-node* fill-in in each *selective block* is considered in SB-BIC(0) [75,77].

```
      do ib= 1, NBLOCKtot
        NB0size= BLOCKstack(ib) - BLOCKstack(ib-1)
        (FORWARD SUBSTITUTIONS)
        do i= 1, NB0size
          ii= i + iBS
          WVAL1= 0.d0
          WVAL2= 0.d0
          WVAL3= 0.d0
          do j= 1, NB0size
            WR1= WKB(3*j-2)
            WR2= WKB(3*j-1)
            WR3= WKB(3*j  )
            WVAL1= WVAL1 + ALU(3*i-2,3*j-2,ib) * WR1                    &
     &                   + ALU(3*i-2,3*j-1,ib) * WR2                    &
     &                   + ALU(3*i-2,3*j  ,ib) * WR3
            WVAL2= WVAL2 + ALU(3*i-1,3*j-2,ib) * WR1                    &
     &                   + ALU(3*i-1,3*j-1,ib) * WR2                    &
     &                   + ALU(3*i-1,3*j  ,ib) * WR3
            WVAL3= WVAL3 + ALU(3*i  ,3*j-2,ib) * WR1                    &
     &                   + ALU(3*i  ,3*j-1,ib) * WR2                    &
     &                   + ALU(3*i  ,3*j  ,ib) * WR3
          enddo
          WW(3*ii-2,Z)= WVAL1
          WW(3*ii-1,Z)= WVAL2
          WW(3*ii  ,Z)= WVAL3
        enddo
      enddo
```

Fig. 4.5 Procedure of the *selective blocking*: Full LU factorization procedure for a (3×NB) × (3×NB) size *selective block.* [75,77]

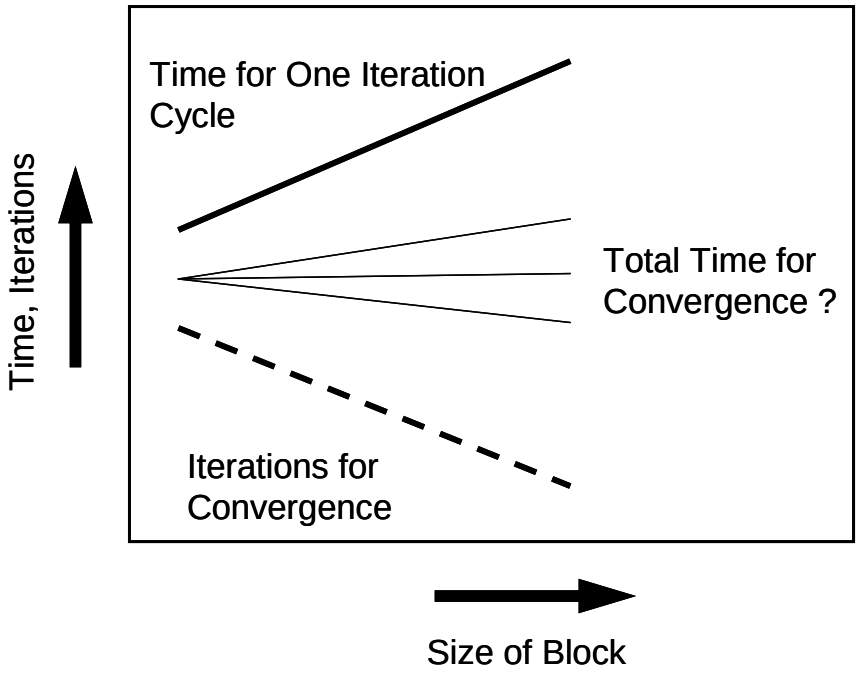


Fig. 4.6 Trade-off between convergence and computational cost per one iteration cycle according to block size in CEBE type method. Based on [18,50,82].

Table 4.2 Iterations/CPU time (includes factorization) for convergence ($\varepsilon=10^{-8}$) on a 4 PE COMPAQ Alpha 21164/600MHz cluster using CG with block preconditioning for the 3D elastic fault-zone contact problem in Fig,4.1 (21,660 DOF). (**ORIGINAL partitioning**)

Preconditioning	λ	Iter #	sec.
BIC(1)	10^0	90	4.1
	10^6	1,724	70.7
BIC(2)	10^0	86	6.6
	10^6	962	59.8
SB-BIC(0)	10^0	156	3.5
	10^6	1,598	33.9

Table 4.3 Iterations/CPU time (includes factorization) for convergence ($\varepsilon=10^{-8}$) on a 4 PE COMPAQ Alpha 21164/600MHz cluster using CG with block preconditioning for the 3D elastic fault-zone contact problem in Fig.4.1 (21,660 DOF). (**IMPROVED partitioning**)

Preconditioning	λ	Iter #	sec.
BIC(1)	10^0	80	3.8
	10^6	167	7.4
BIC(2)	10^0	71	5.8
	10^6	74	5.9
SB-BIC(0)	10^0	126	2.9
	10^6	124	2.8

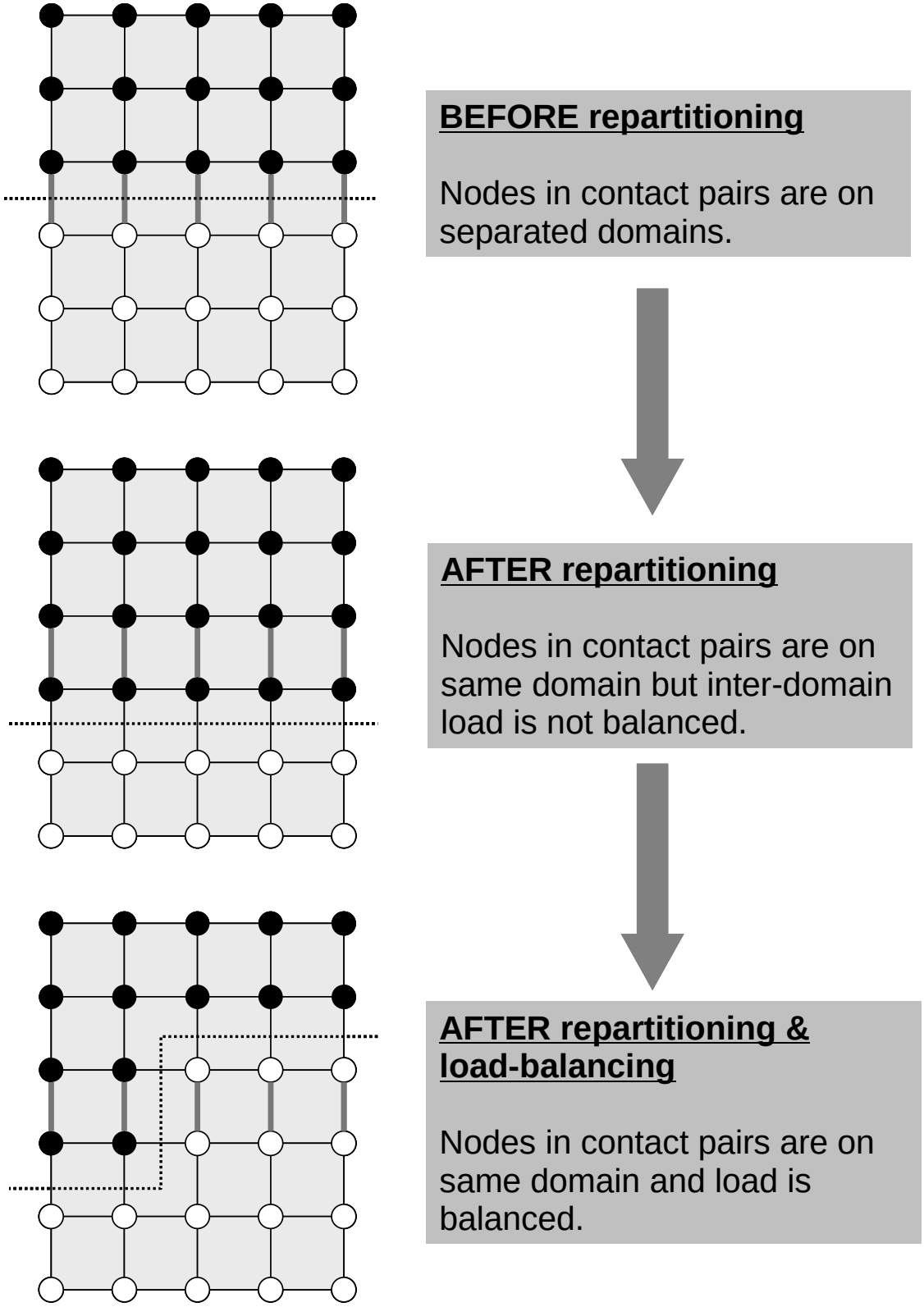
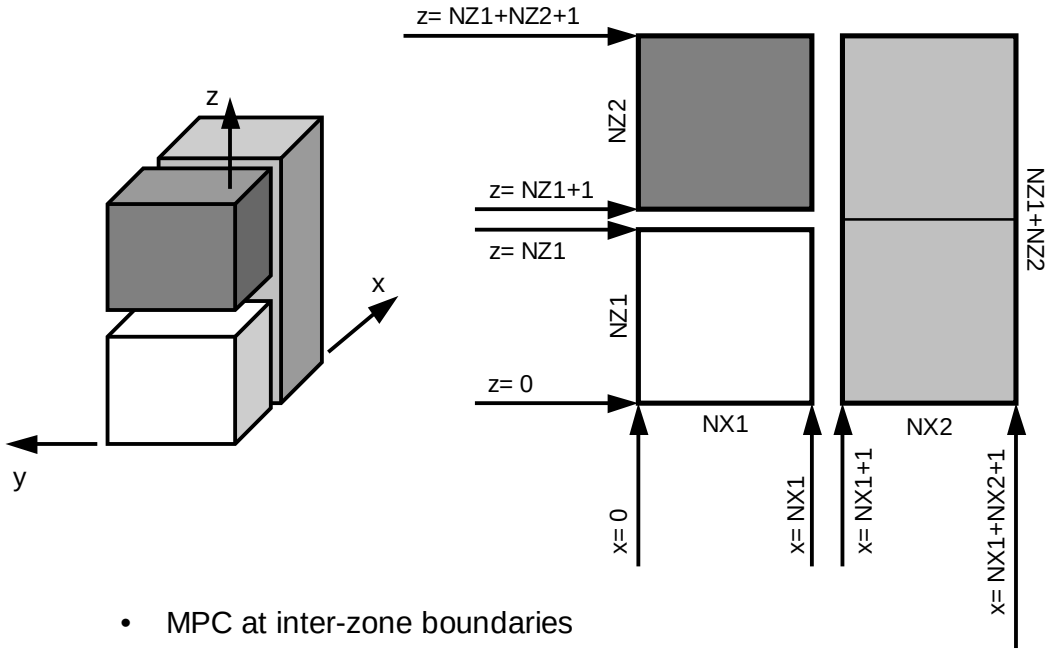
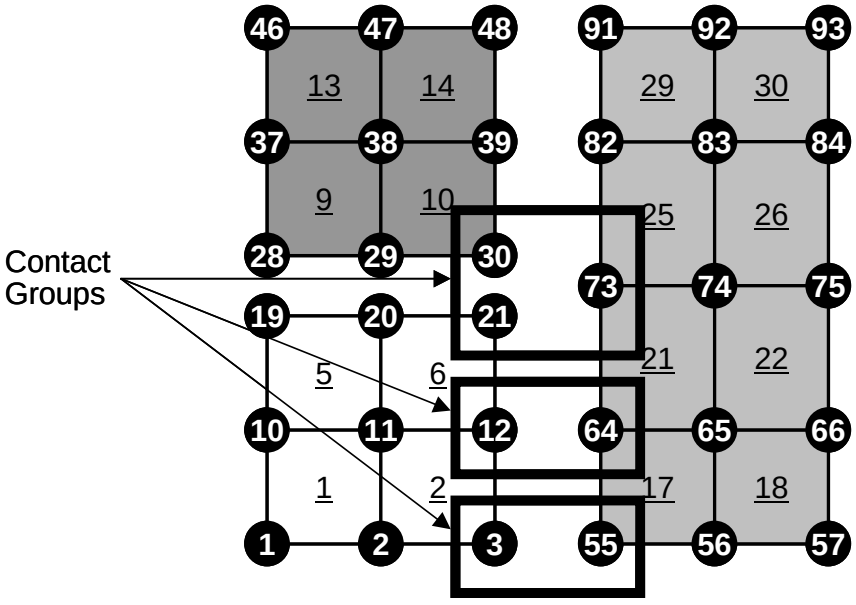


Fig. 4.7 Partitioning strategy for the nodes in contact groups [75,77]



- MPC at inter-zone boundaries
- Symmetric condition at the x=0 and y=0 surfaces
- Dirichlet fixed condition at the z=0 surface
- Uniform distributed load at the z= Zmax surface

(a) Model and boundary conditions



(b) Node, elements and contact groups

Fig. 4.8 Description of the simple block model [75,77]

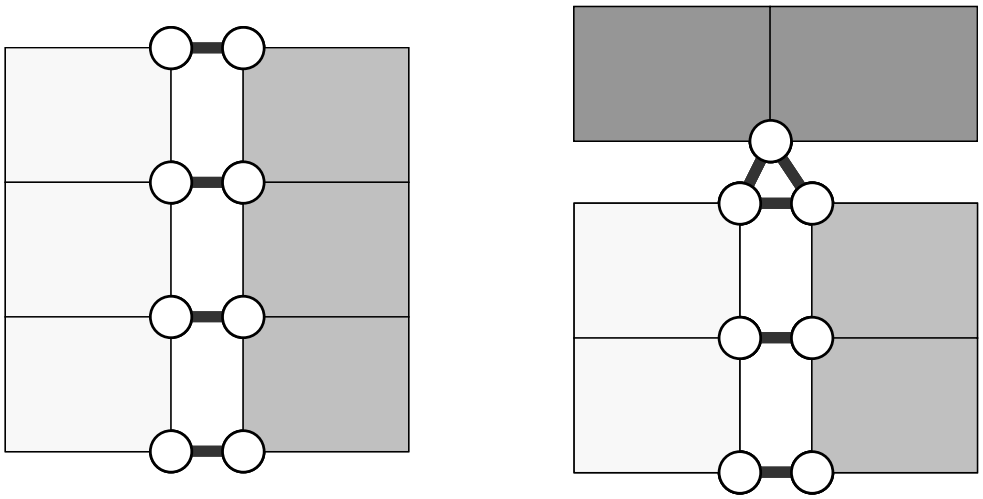


Fig.4.9 111-type element (Rod/Beam) in GeoFEM is put in each contact group and very large stiffness corresponding to penalty is applied [75,77]

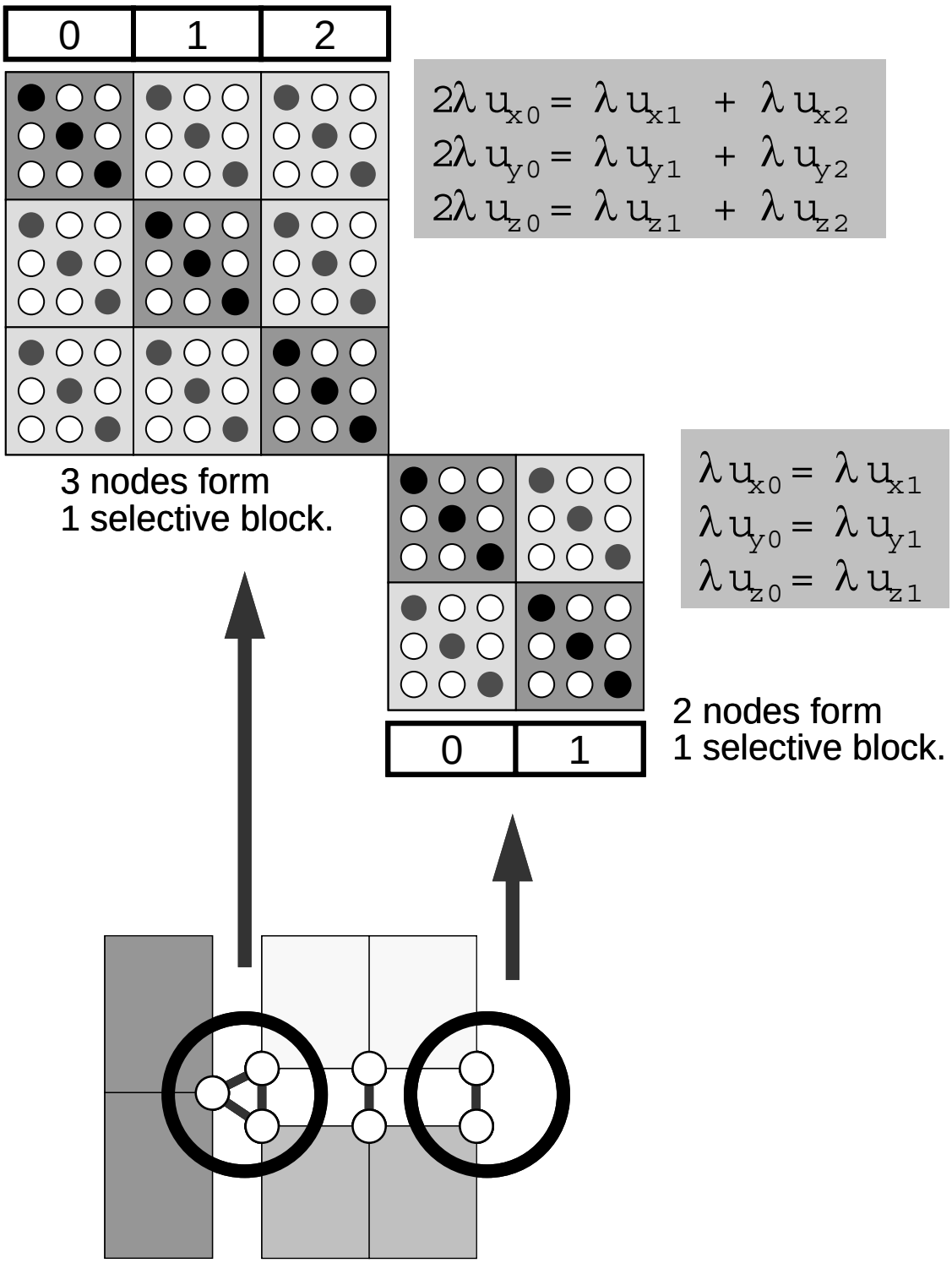


Fig.4.10 Matrix operation of nodes in a contact group [75,77]

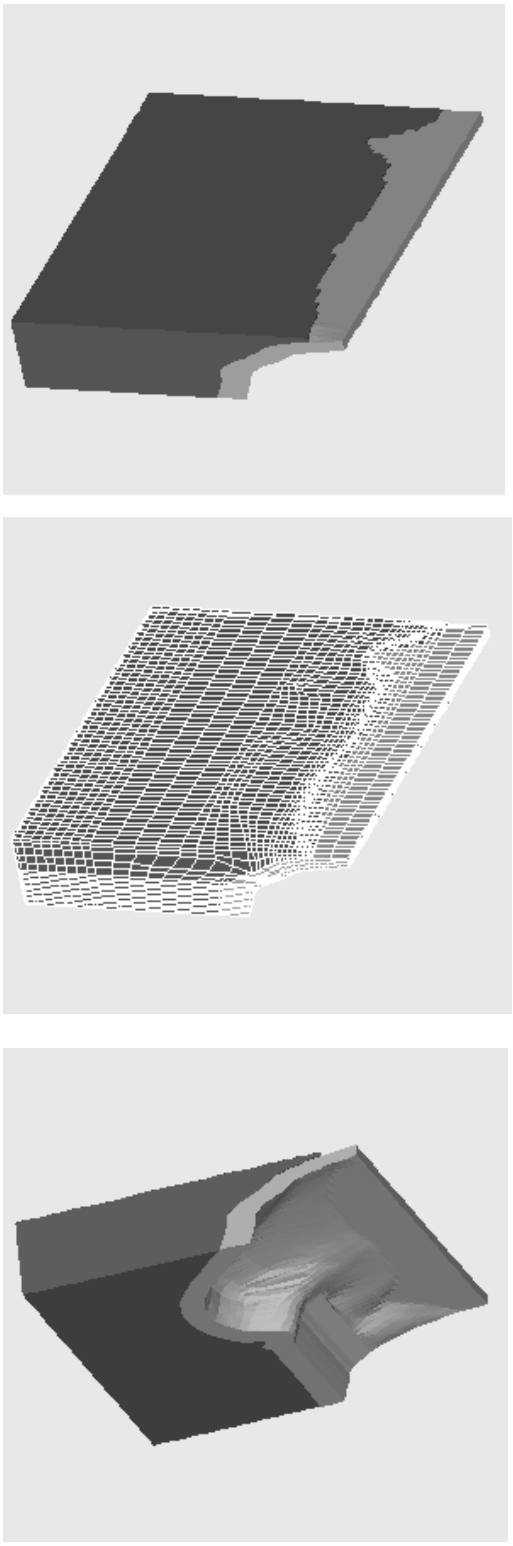


Fig. 4.11 Description of the Southwest Japan model This model consists crust (dark gray) and subduction plate (light gray). 27,195 nodes and 23,831 tri-linear (1st order) hexahedral elements are included [131].

Table 4.4 Iterations/CPU time (includes factorization) for convergence ($\varepsilon=10^{-8}$) on a single PE of COMPAQ Alpha 21164/600MHz by preconditioned CG for the 3D elastic contact problem for simple block model with MPC condition in Fig.4.8 (83,664DOF).: **BIC(n)**: Block IC with n-level fill-in, **SB-BIC(0)**: BIC(0) with the selective blocking reordering.

Preconditioning	λ	Iter #	sec.
BIC(0)	10^2	388	202.
	10^4	No Conv.	N/A
BIC(1)	10^2	77	89.
	10^6	77	89.
	10^{10}	78	90.
BIC(2)	10^2	59	135.
	10^6	59	135.
	10^{10}	60	137.
SB-BIC(0)	10^2	114	61.
	10^6	114	61.
	10^{10}	114	61.

Table 4.5 Largest and smallest eigenvalues (E_{min}, E_{max}) and $\kappa = E_{max}/E_{min}$ of $[M]^{-1}[A]$ for a wide range of penalty parameter values: 3D elastic contact problem for simple block model with MPC condition in Fig.4.8 (83,664DOF).

Preconditioning		$\lambda=10^2$	$\lambda=10^6$	$\lambda=10^{10}$
BIC(0)	E_{min}	4.845568E-03	4.865363E-07	4.865374E-11
	E_{max}	1.975620E+00	1.999998E+00	2.000000E+00
	κ	4.077170E+02	4.110686E+06	4.110681E+10
BIC(1)	E_{min}	8.901426E-01	8.890643E-01	8.890641E-01
	E_{max}	1.013930E+00	1.013863E+00	1.013863E+00
	κ	1.139065E+00	1.140371E+00	1.140371E+00
BIC(2)	E_{min}	9.003662E-01	8.992896E-01	8.992895E-01
	E_{max}	1.020256E+00	1.020144E+00	1.020144E+00
	κ	1.133157E+00	1.134388E+00	1.134389E+00
SB-BIC(0)	E_{min}	6.814392E-01	6.816873E-01	6.816873E-01
	E_{max}	1.005071E+00	1.005071E+00	1.005071E+00
	κ	1.474924E+00	1.474387E+00	1.474387E+00

Table 4.6 Iterations/CPU time (includes factorization) for convergence ($\varepsilon=10^{-8}$) on a single PE of COMPAQ Alpha 21164/600MHz by preconditioned CG for the 3D elastic contact problem for Southwestern Japan model with MPC condition in Fig.4.11 (81,585DOF).: **BIC(n)**: Block IC with n-level fill-in, **SB-BIC(0)**: BIC(0) with the selective blocking reordering.

Preconditioning	λ	Iter #	sec.
BIC(0)	10^2	344	172.
	10^4	No Conv.	N/A
BIC(1)	10^2	201	192.
	10^4	256	237.
	10^6	256	237.
	10^8	258	240.
	10^{10}	259	241.
BIC(2)	10^2	176	288.
	10^4	229	360.
	10^6	230	361.
	10^8	230	361.
	10^{10}	232	364.
SB-BIC(0)	10^2	297	149.
	10^4	295	148.
	10^6	295	148.
	10^8	295	148.
	10^{10}	295	148.

Table 4.7 Largest and smallest eigenvalues (E_{min}, E_{max}) and $\kappa = E_{max}/E_{min}$ of $[M]^{-1}[A]$ for a wide range of penalty parameter values: 3D elastic contact problem for Southwestern Japan model with MPC condition in Fig.4.11 (81,585DOF).

Preconditioning		$\lambda=10^2$	$\lambda=10^4$	$\lambda=10^6$	$\lambda=10^{10}$
BIC(0)	E_{min}	1.970395E-02	1.999700E-04	1.999997E-06	2.000000E-10
	E_{max}	1.005194E+00	1.005194E+00	1.005194E+00	1.005194E+00
	κ	5.101486E+01	5.026725E+03	5.025979E+05	5.025971E+09
BIC(1)	E_{min}	3.351178E-01	2.294832E-01	2.286390E-01	2.286306E-01
	E_{max}	1.142246E+00	1.142041E+00	1.142039E+00	1.142039E+00
	κ	3.408491E+00	4.976580E+00	4.994944E+00	4.995128E+00
BIC(2)	E_{min}	3.558432E-01	2.364909E-01	2.346180E-01	2.345990E-01
	E_{max}	1.058883E+00	1.088397E+00	1.089189E+00	1.089196E+00
	κ	2.975702E+00	4.602277E+00	4.642391E+00	4.642800E+00
SB-BIC(0)	E_{min}	2.380572E-01	2.506369E-01	2.507947E-01	2.507963E-01
	E_{max}	1.005194E+00	1.005455E+00	1.005465E+00	1.005466E+00
	κ	4.222491E+00	4.011600E+00	4.009117E+00	4.009092E+00

Table 4.8 Iterations/elapsed execution time (includes factorization, communication overhead) for convergence ($\varepsilon=10^{-8}$) on a Hitachi SR2201 with 128 PEs using preconditioned CG for the 3D elastic contact problem for simple block model with MPC condition in Fig.4.8 (2,471,439 DOF). Domains are partitioned according to the contact group information.: **BIC(n)**: Block IC with n-level fill-in, **SB-BIC(0)**: BIC(0) with the selective blocking reordering.

Preconditioning	λ	Iter #	sec.
BIC(0)	10^2	998	118.
	10^4	No Conv.	N/A
BIC(1)	10^2	419	98.
	10^6	419	98.
	10^{10}	421	99.
BIC(2)	10^2	394	171.
	10^6	394	171.
	10^{10}	396	172.
SB-BIC(0)	10^2	565	71.
	10^6	566	71.
	10^{10}	567	72.

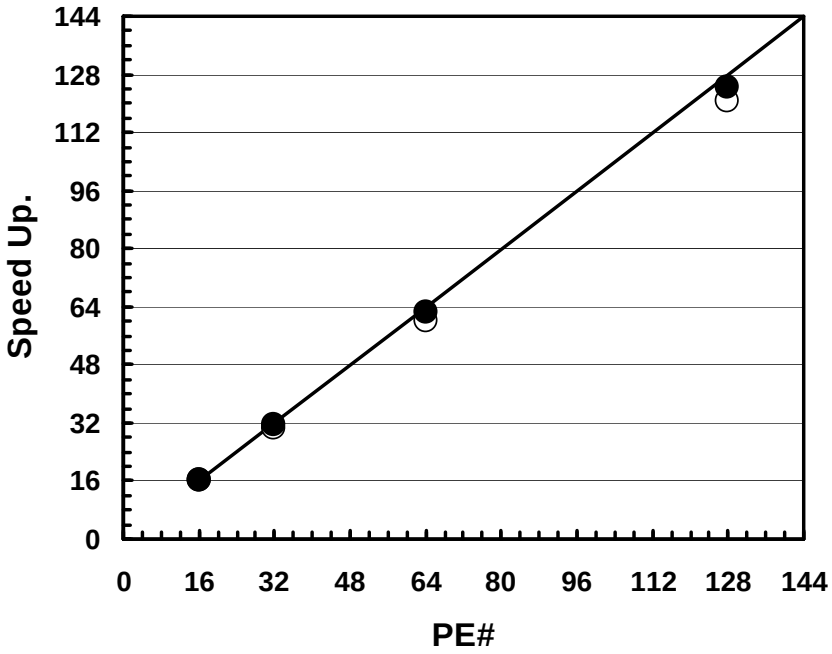
Table 4.9 Iterations/elapsed execution time (includes factorization, communication overhead) for convergence ($\varepsilon=10^{-8}$) on a Hitachi SR2201 with 16 to 128 PEs using preconditioned CG for the 3D elastic contact problem for simple block model with MPC condition in Fig.4.8 (2,471,439 DOF). Domains are partitioned according to the contact group information.: **BIC(n)**: Block IC with n-level fill-in, **SB-BIC(0)**: BIC(0) with the selective blocking reordering.

$\lambda=10^2$

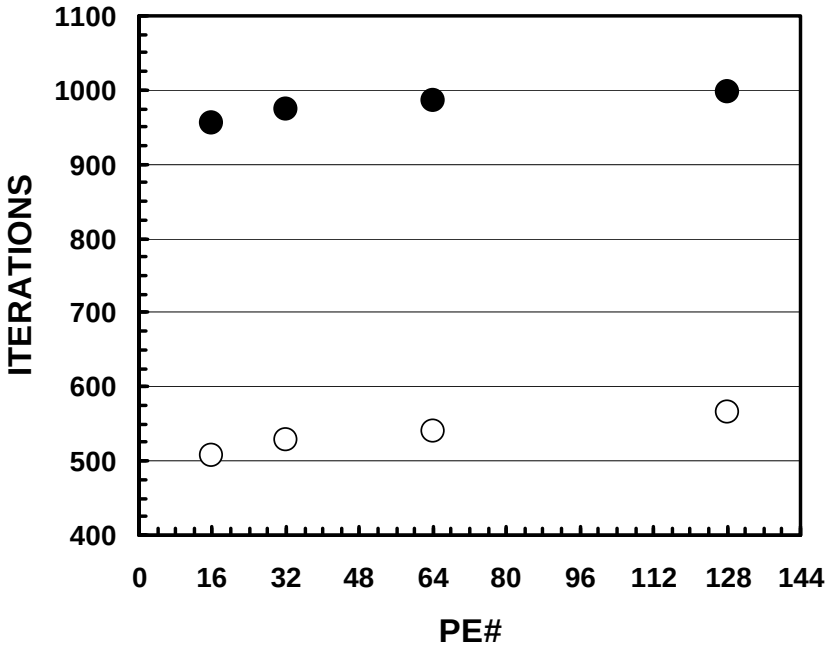
Preconditioning		16 PEs	32 PEs	64 PEs	128 PEs
BIC(0)	iters	956	975	986	998
	sec.	919	469	236	118
	ratio	16.0	31.4	62.5	124.4
BIC(1)	iters			396	419
	sec.	N/A	N/A	190	98
	ratio			64.0	124.3
BIC(2)	iters				394
	sec.	N/A	N/A	N/A	171
	ratio				-
SB-BIC(0)	iters	508	529	541	565
	sec.	540	282	144	71
	ratio.	16.0	30.7	60.2	120.7

$\lambda=10^6$

Preconditioning		16 PEs	32 PEs	64 PEs	128 PEs
BIC(1)	iters			395	419
	sec.	N/A	N/A	190	98
	ratio			64.0	124.2
BIC(2)	iters				394
	sec.	N/A	N/A	N/A	171
	ratio				-
SB-BIC(0)	iters	510	532	543	566
	sec.	542	283	144	71
	ratio	16.0	30.6	60.5	121.9



(a) Speed-up ratio



(b) Iteration number for convergence

Fig. 4.12 Parallel performance based on elapsed execution time including communication and iterations for convergence ($\varepsilon=10^{-8}$) on a Hitachi SR2201 with 16 to 128 PEs using preconditioned CG for the 3D elastic contact problem with MPC condition ($\lambda=10^{2}$) in Fig.4.8 (2,471,439 DOF). Domains are partitioned according to the contact group information. (White Circles: SB-BIC(0), Black-Circles: BIC(0)).

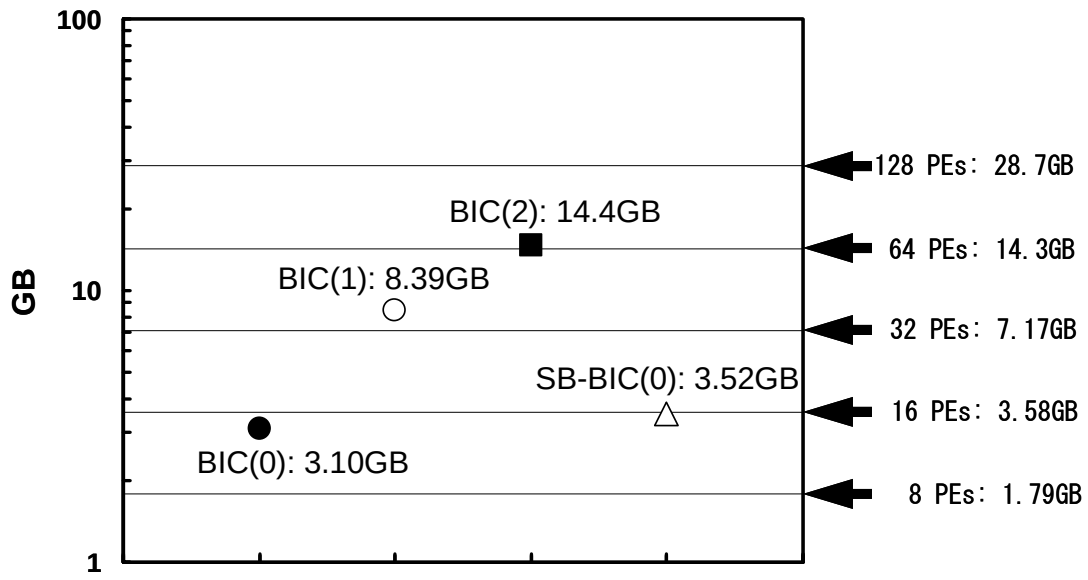
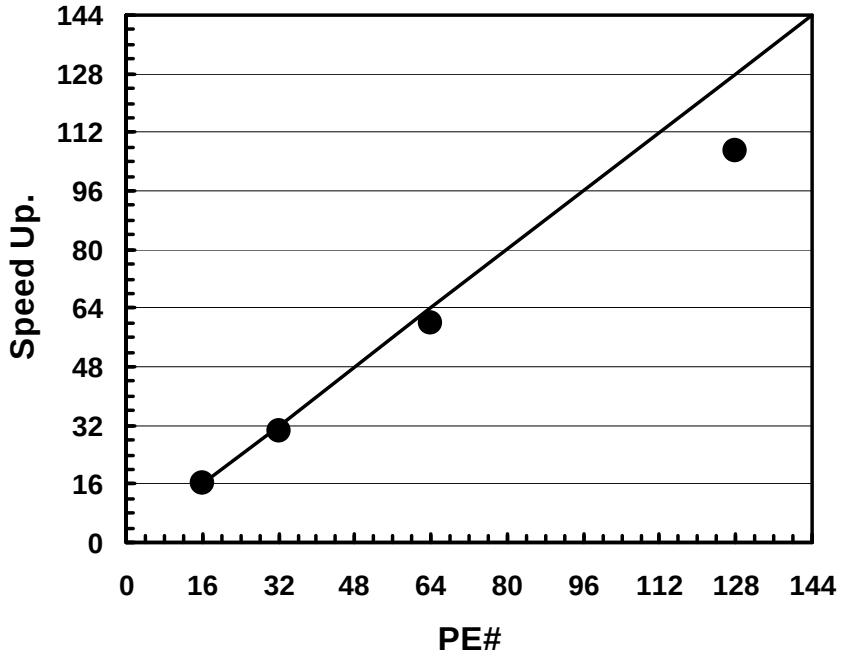


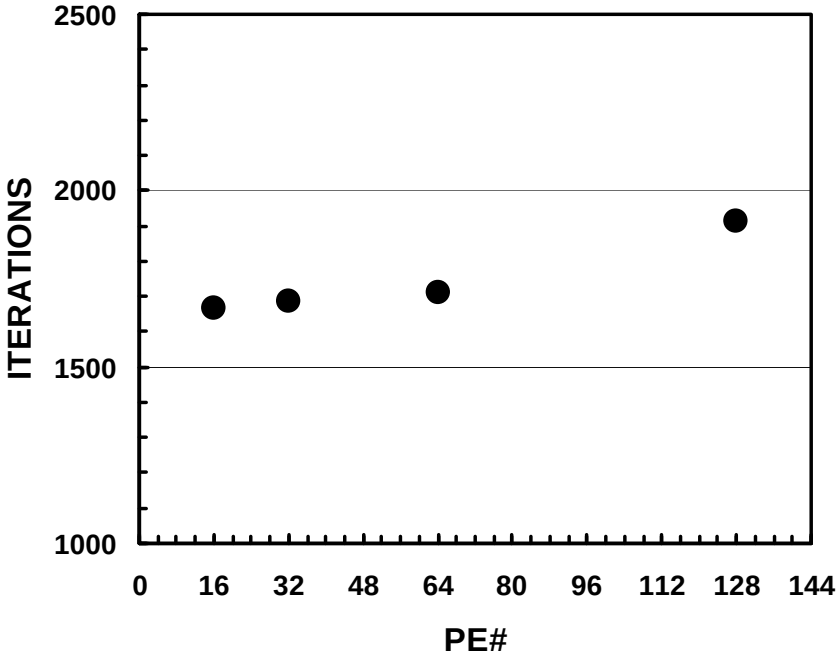
Fig. 4.13 Required memory size of CG solvers with various types preconditioners for the 3D elastic contact problem with MPC condition ($\lambda=10^2$) in Fig.4.8 (2,471,439 DOF) and available memory size on Hitachi SR2201 (Black-Circles: BIC(0), White-Circles: BIC(1), Black-Squares: BIC(2), White Triangles: SB-BIC(0)).

Table 4.10 Iterations/elapsed execution time (including factorization, communication overhead) for convergence ($\varepsilon=10^{-8}$) on a Hitachi SR2201 with 16 to 128 PEs using SB-BIC(0) CG for the 3D elastic contact problem for Southwest Japan model with MPC condition ($\lambda=10^6$) in Fig.4.11 (2,992,264 DOF). Domains are partitioned according to the contact group information.

Preconditioning		16 PEs	32 PEs	64 PEs	128 PEs
SB-BIC(0)	iters	1665	1686	1710	1912
	sec.	1901.	993.	506.	284.
	ratio	16.0	30.6	60.1	107.2



(a) Speed-up ratio



(b) Iteration number for convergence

Fig. 4.14 Parallel performance based on elapsed execution time including communication and iterations for convergence ($\varepsilon=10^{-8}$) on a Hitachi SR2201 with 16 to 128 PEs using SB-BIC(0) CG for the 3D elastic contact problem with MPC condition ($\lambda=10^{6}$) in Fig.4.11 (2,992,266 DOF). Domains are partitioned according to the contact group information.

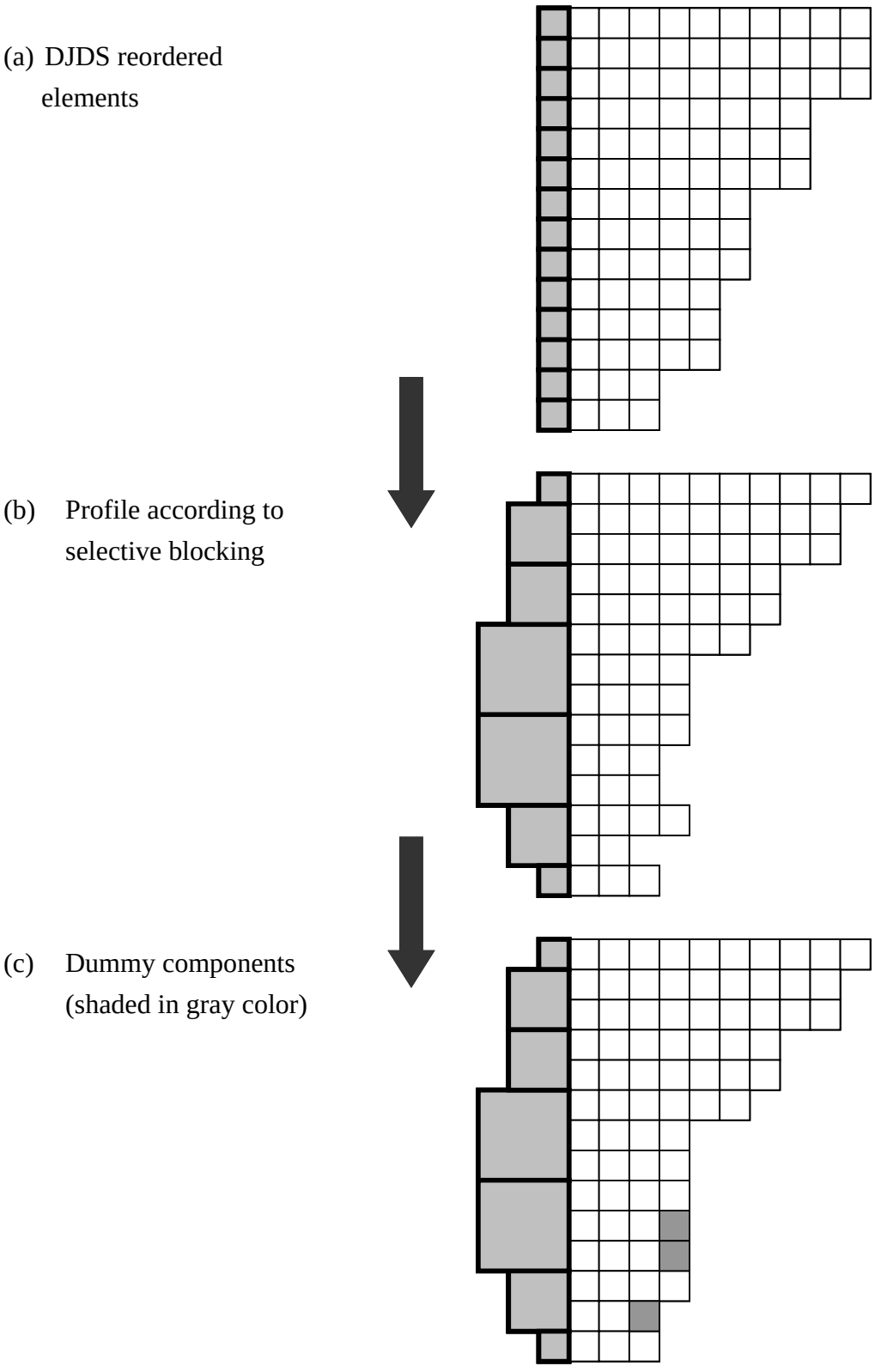


Fig. 4.15 Dummy elements to maintain a smooth decrease in the number of off-diagonal components in descending order

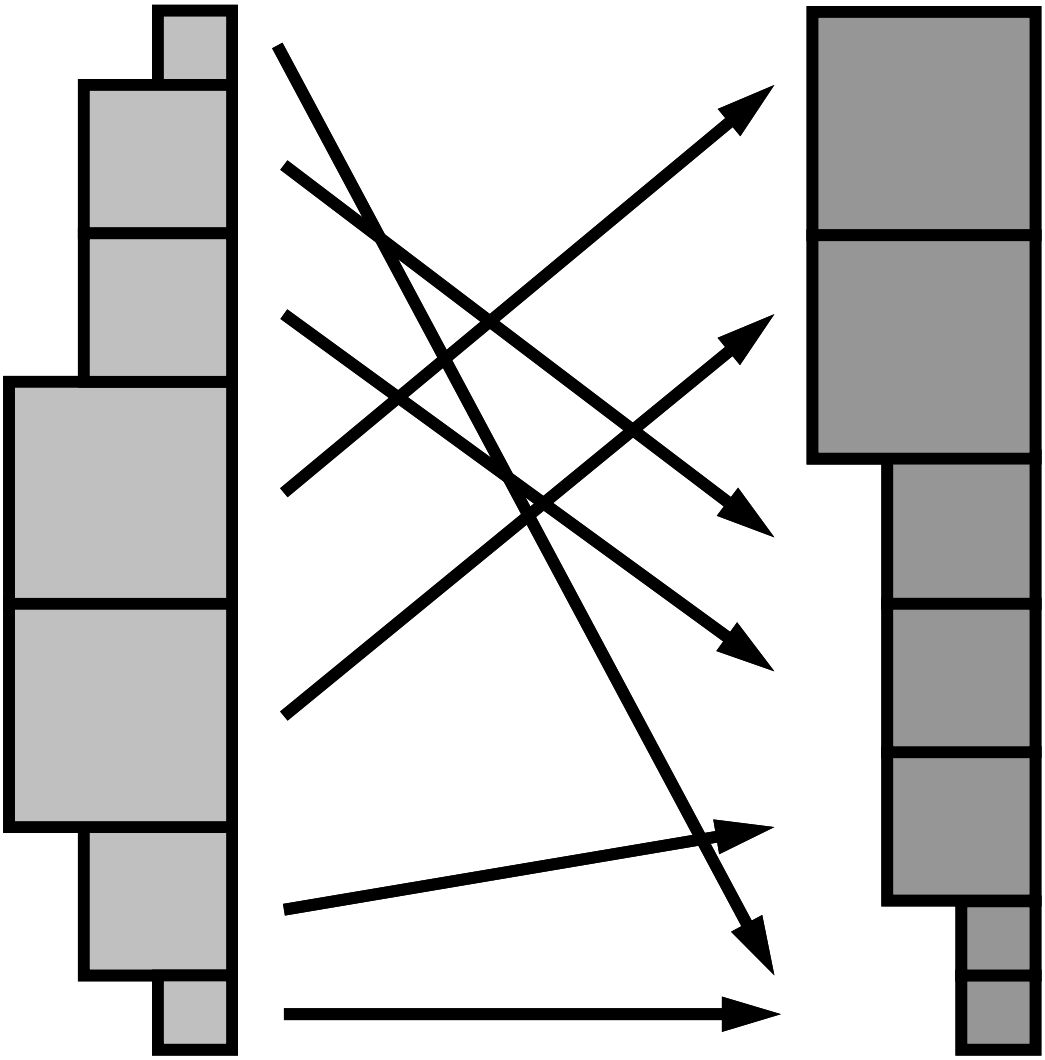
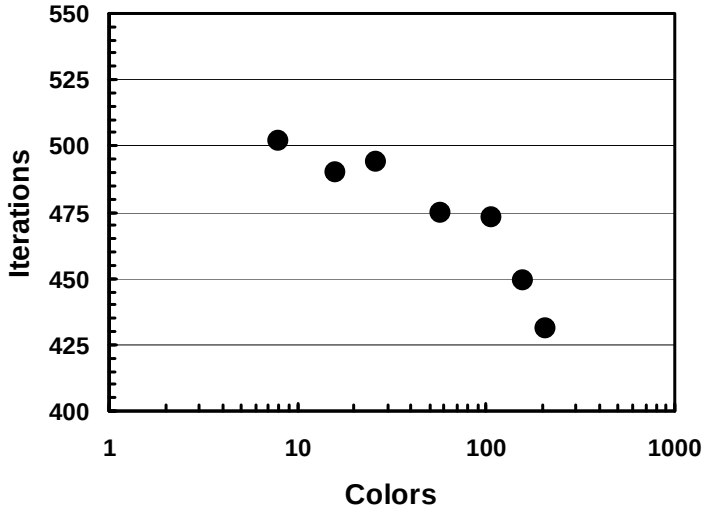
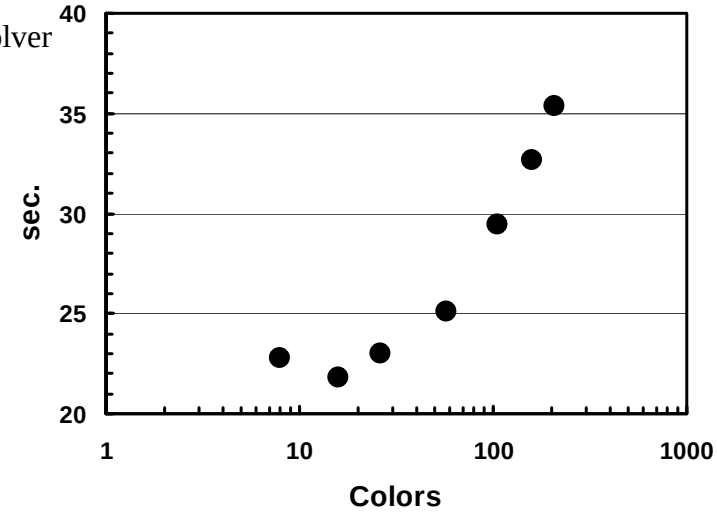


Fig. 4.16 Reordering of selective blocks (supernodes) according to block size.

(a) Iterations for convergence



(b) Elapsed time for the linear solver



(c) GFLOPS rate

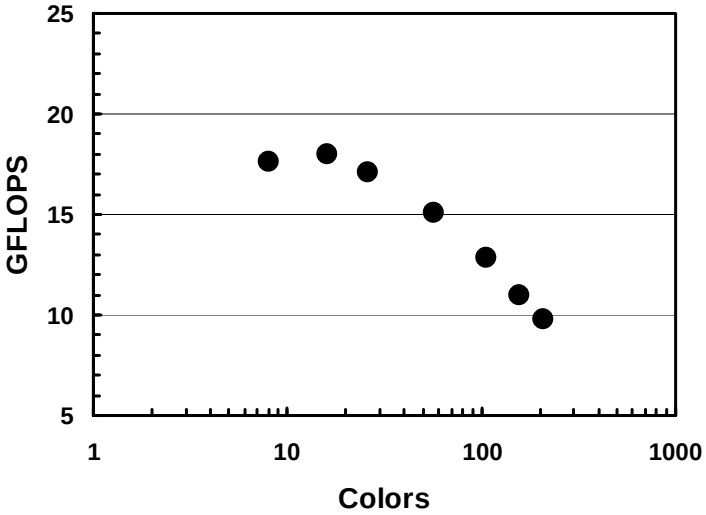
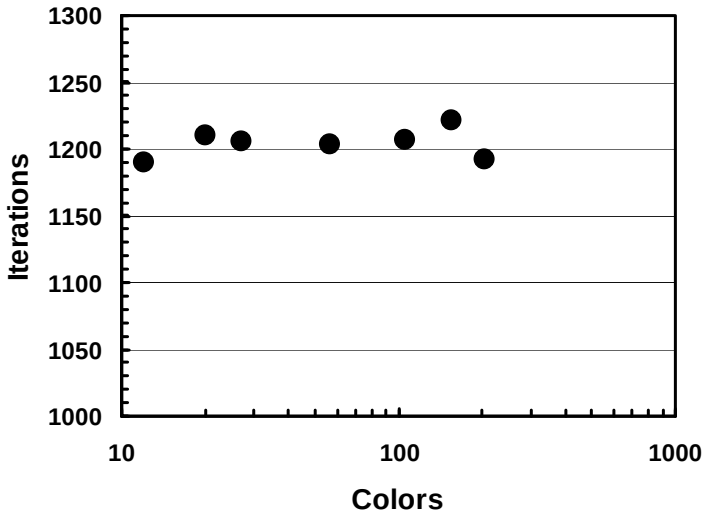
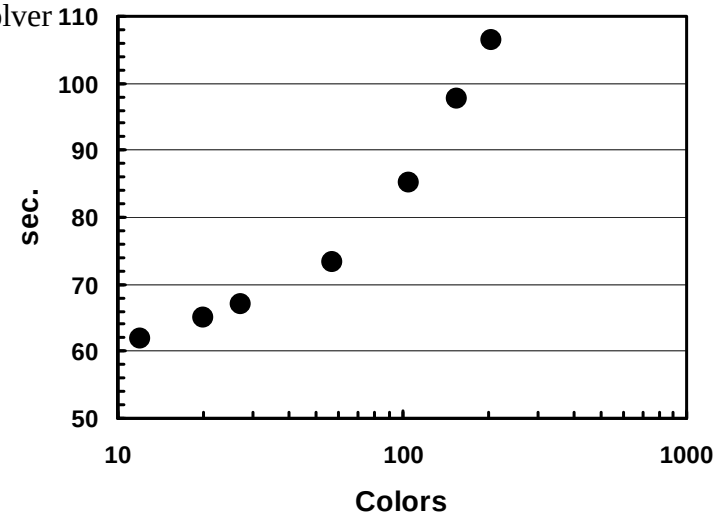


Fig. 4.17 Performance on a single SMP node of the Earth Simulator (peak performance = 64GFLOPS) using SB-BIC(0) CG for the 3D elastic contact problem with MPC condition ($\lambda=10^6$) in Fig.4.8 (Simple Block Model). (a) Iterations for convergence, (b) Elapsed time for the linear solver, and (c) GFLOPS rate.

(a) Iterations for convergence



(b) Elapsed time for the linear solver



(c) GFLOPS rate

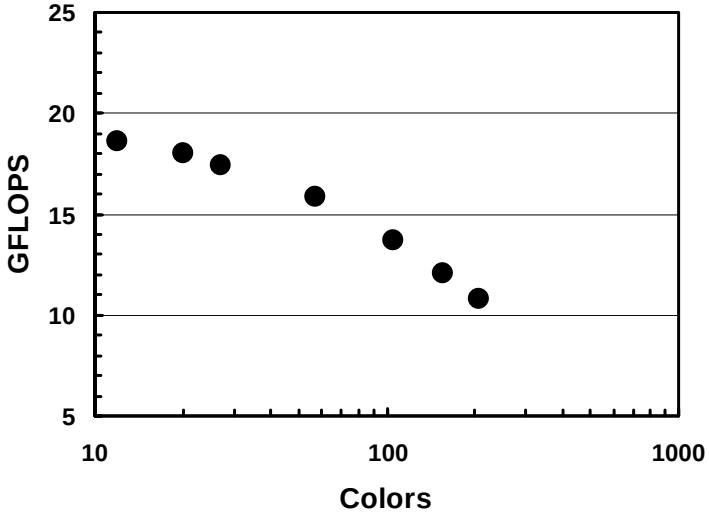
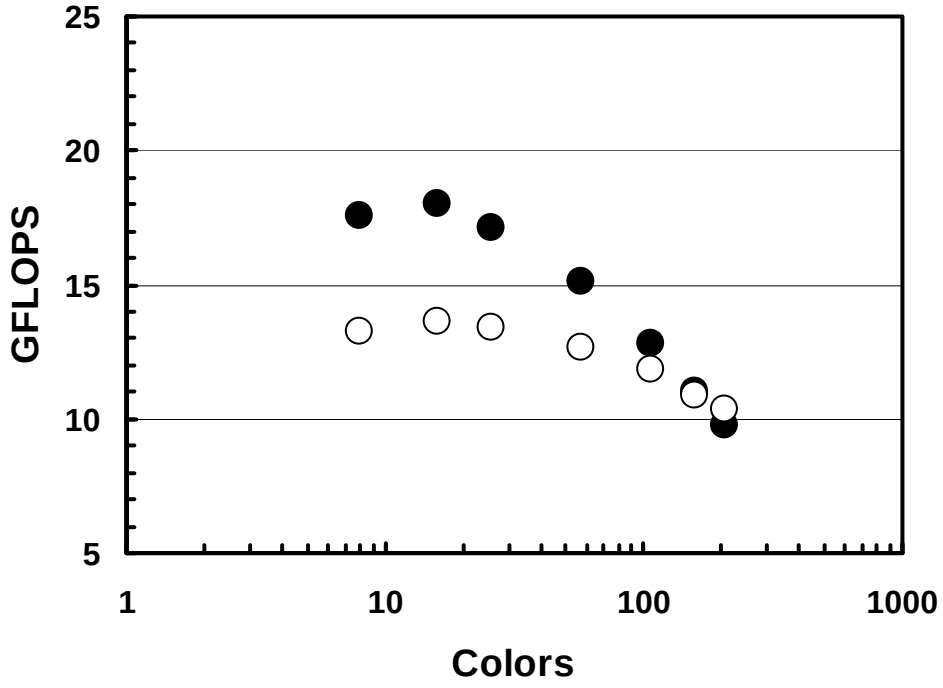


Fig. 4.18 Performance on a single SMP node of the Earth Simulator (peak performance = 64GFLOPS) using SB-BIC(0) CG for the 3D elastic contact problem with MPC condition ($\lambda=10^6$) in Fig.4.11 (Southwest Japan model). (a) Iterations for convergence, (b) Elapsed time for the linear solver, and (c) GFLOPS rate.

(a) Simple Block



(b) Southwest Japan

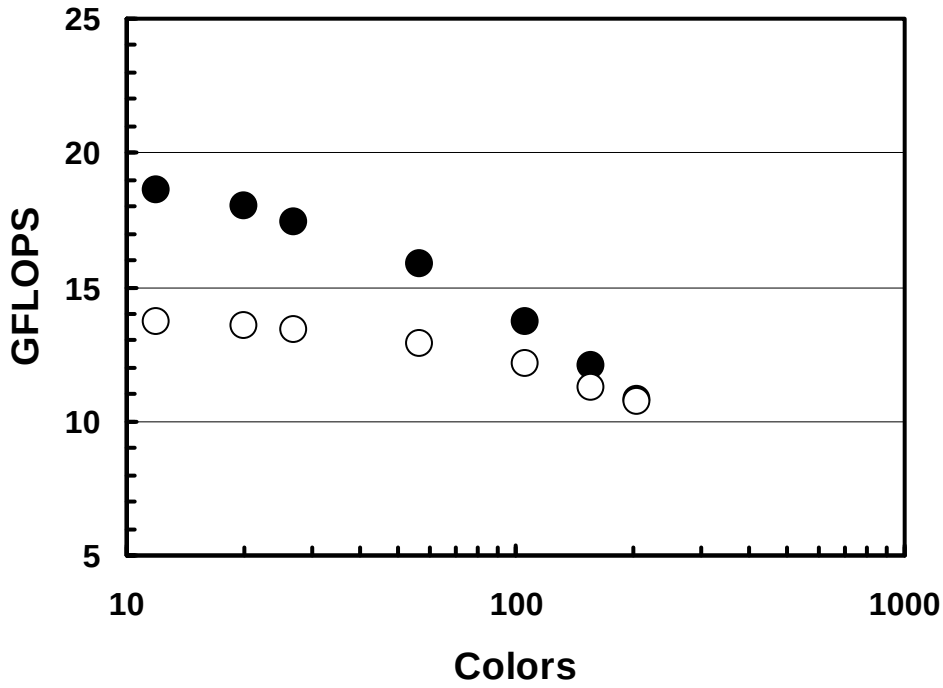
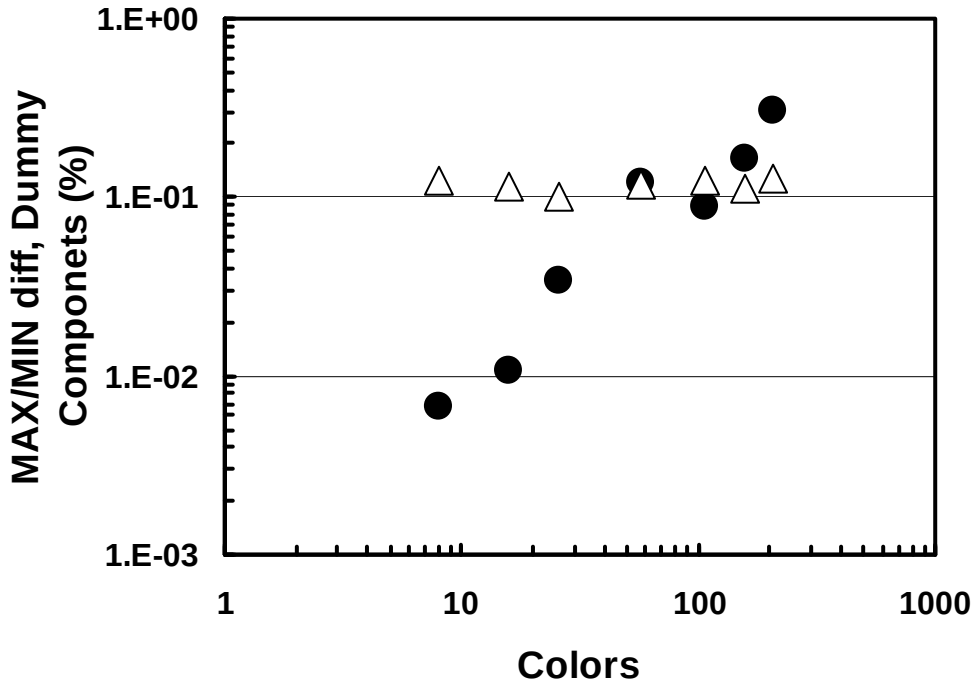


Fig. 4.19 Effect of reordering of selective block on a single SMP node of the Earth Simulator (peak performance = 64GFLOPS) using SB-BIC(0) CG for the 3D elastic contact problem with MPC condition ($\lambda=10^6$). (a) Simple Block (b) Southwest Japan. (BLACK Circles: WITH reordering, WHITE Circles: WITHOUT reordering).

(a) Simple Block



(b) Southwest Japan

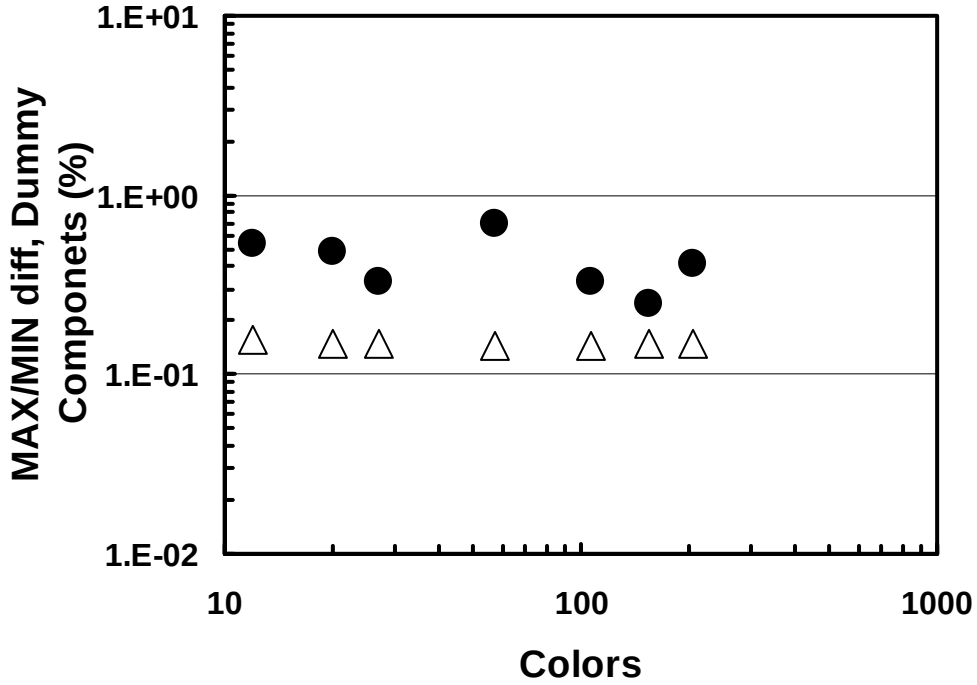


Fig. 4.20 Load imbalance on a single SMP node for selective blocking preconditioning. (a) Simple Block (b) Southwest Japan. (BLACK Circles: Load-imbalance among PEs on the SMP node, WHITE Triangles: Ratio of dummy off-diagonal components).

Chapter 5 Parallel Multilevel Iterative Linear Solvers with Unstructured Adaptive Grids

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 chapter, 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. Extended local data structure based on that of GeoFEM has been developed for multilevel parallel procedure. Computations using this method on a Hitachi SR2201 with up to 128 processors by flat MPI parallel programming model demonstrated good scalability.

5.1 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) [7] 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 [71,72,73,79,81,84,131] 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 [10,11,110] 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 [1,2,107,109,110,112,128], for both parallel and serial computers.

In this study, a multigrid-preconditioned conjugate gradient iterative method (MGCG) on parallel computers was developed with local data structure for multilevel parallel procedure based on that of GeoFEM and applied to Poisson equations in the region between two spherical surfaces on adaptively generated semi-unstructured prismatic grids based on the method in [42,76,89,91]. This procedure has also been applied to grids with local refinement.

5.2 Incompressible Navier-Stokes Method

5.2.1 Background

In this chapter, the target application is 3D incompressible thermal convection in the region between two 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). Entire regions is discretized by prismatic grids.

Incompressible flows are frequently encountered in engineering applications. During the past three decades a significant number of numerical algorithms have been developed for solution of the incompressible Navier-Stokes equations [30,43,68,69,70]. The lack of pressure term in the continuity equation makes solution of the momentum equations with the divergence-free constraint more difficult. In case of incompressible flows, the conservation of mass acts as a constraint condition that the velocity field needs to satisfy, while in compressible flows, the conservation of mass is given as a partial differential equation for the temporal variation of density. The infinite speed of sound in the incompressible case requires an implicit treatment for the pressure. Furthermore, the incompressibility constraint may produce an oscillatory pressure field.

In this work, pressure correction scheme with a special Poisson equation for the pressure field [33,92,93,99] was adopted. The usual computational procedure is to assume an initial pressure field, and then an iterative process is defined until the continuity equation is satisfied.

A major issue with pressure and velocity spatial discretization is oscillations in the pressure field. In order to suppress these modes, *staggered* grids have been employed by several of these algorithms. The pressure field is stored at different locations from the velocity values [33,92,93]. On the other hand, employment of non-staggered grids [27,32,43,49,55,56,57,68,69,70,94,101,108] requires dissipation in the algorithms. Stability of both approaches with high Reynolds number flows is an important issue. In this work, staggered cell was adopted. Pressure and potential for pressure correction are defined at the cell centers, while the velocity components and temperature are defined at cell corners (Fig.5.1).

5.2.2 Governing Equations and Pressure Correction Scheme

The non-dimensionalized Navier-Stokes equations of three-dimensional laminar incompressible flow with thermal convection are presented. The pressure correction formulation based on the SMAC scheme is also described. Boundary conditions for the

momentum equation and pressure correction equation are presented.

The governing equations are the non-dimensionalized Navier-Stokes equations of three-dimensional laminar incompressible flow with thermal convection:

$$\nabla \cdot \mathbf{u} = 0 \tag{5.1}$$

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot (\mathbf{uu}) + \nabla p - \frac{1}{Re} \Delta \mathbf{u} + \mathbf{g} \frac{Gr}{Re^2} T = 0 \tag{5.2}$$

$$\frac{\partial T}{\partial t} + \nabla \cdot (\mathbf{u}T) - \frac{1}{Re\,Pr} \Delta T = Q \tag{5.3}$$

where:

$\mathbf{u}$	velocity vector ($= (u,v,w)$)
p	pressure
T	temperature
Re	Reynolds number ($= \frac{\rho U L}{\mu}$, where ρ is reference velocity, U is reference velocity, L is reference length and μ is viscosity)
Gr	Grashof number ($= \frac{g\beta \Delta T L^3}{\mu^2}$, where g is gravitational acceleration, β is coefficient of thermal expansion and ΔT is reference temperature difference)
Pr	Prandtl number ($= \nu/a$, where n is kinematic viscosity ($=\mu/\rho$) and a is thermal diffusivity)

Equation (5.1) is the continuity equation, (5.2) is the momentum equation and (5.3) is the energy equation.

An explicit/implicit scheme is adopted for integration in time of the above equations. The velocity and temperature values are treated explicitly, while the pressure values are treated implicitly in the momentum equations. The velocity and temperature values are marched in time with a forward Euler scheme [33]. The continuity equation is formulated implicitly with the velocity values considered at time level (n+1).
Specifically, the corresponding semi-discrete system is written as follows:

$$\nabla \cdot \mathbf{u}^{(n)} = 0 \tag{5.4}$$

$$\frac{\mathbf{u}^{(n+1)} - \mathbf{u}^{(n)}}{\Delta t} + \nabla \cdot \left(\mathbf{u}^{(n)}\mathbf{u}^{(n)}\right) + \nabla p^{(n+1)} - \frac{1}{Re}\Delta \mathbf{u}^{(n)} + \mathbf{g}\frac{Gr}{Re^2}T^{(n)} = 0 \quad (5.5)$$

$$\frac{T^{(n+1)} - T^{(n)}}{\Delta t} + \nabla \cdot \left(\mathbf{u}^{(n+1)}T^{(n)}\right) - \frac{1}{Re\,Pr}\Delta T^{(n)} = Q^{(n)} \quad (5.6)$$

where the superscripts denote the time levels.

The above equation cannot be solved directly due to the implicit treatment of the pressure term. An auxiliary velocity vector **u'** is introduced, which satisfies the following equation:

$$\frac{\mathbf{u}' - \mathbf{u}^{(n)}}{\Delta t} + \nabla \cdot \left(\mathbf{u}^{(n)}\mathbf{u}^{(n)}\right) + \nabla p^{(n)} - \frac{1}{Re}\Delta \mathbf{u}^{(n)} + \mathbf{g}\frac{Gr}{Re^2}T^{(n)} = 0 \quad (5.7)$$

In this equation, the pressure term is treated explicitly and **u'** can be obtained directly. However, the solution **u'** does not satisfy the continuity equation.
Subtracting equation (5.7) from (5.5), it is obtained:

$$\mathbf{u}^{(n)} - \mathbf{u}' = -\left[\nabla\left(p^{(n+1)} - p^{(n)}\right)\right]\Delta t \quad (5.8)$$

Introducing a scalar potential ϕ, such that

$$\mathbf{u}^{(n)} - \mathbf{u}' = -\nabla\phi \quad (5.9)$$

the following equation for pressure can be obtained:

$$p^{(n+1)} - p^{(n)} = \frac{1}{\Delta t}\phi \quad (5.10)$$

Finally, taking the divergence of each side of equation (5.9) and considering the continuity equation (5.4), the following Poisson equation is obtained:

$$\Delta\phi = \nabla \cdot \mathbf{u}' \quad (5.11)$$

This equation requires a linear solver. Using ϕ obtained by the above equation, we can

correct the velocity and pressure fields using equations (.9) and (.10) as follows:

$$\mathbf{u}^{(n)} = \mathbf{u}' - \nabla\phi \tag{5.12}$$

$$p^{(n+1)} = p^{(n)} + \frac{1}{\Delta t}\phi \tag{5.13}$$

The above solution procedure follows the SMAC method [46]. The overall solution procedure corresponding to marching by one-time-step is summarized as follows:

(1) calculate the auxiliary velocity vector $\mathbf{u}'$ by (5.7) using $\mathbf{u}^{(n)}$ and $p^{(n)}$ values.
(2) solve (5.11) using $\mathbf{u}^{(n)}$ by linear solver and obtain ϕ values.
(3) calculate $\mathbf{u}^{(n+1)}$ and $p^{(n+1)}$ using (5.12) and (5.13).
(4) if $\nabla \cdot \mathbf{u}^{(n+1)} < \varepsilon$ where ε is the tolerance for divergence, calculate the temperature field by solving (5.6) explicitly using velocity field $\mathbf{u}^{(n+1)}$ and advance to next time step. if not, consider $\mathbf{u}^{(n+1)}$ as $\mathbf{u}'$ and repeat step 2 and 3.

There are two types of boundary conditions employed for the velocity field. These conditions on the two types boundaries, $\partial\Omega_1$ and $\partial\Omega_2$ are as follows:

$$\begin{aligned} &\text{on } \partial\Omega_1 && \mathbf{u} = \mathbf{u_0} \\ &\text{on } \partial\Omega_2 && -p\mathbf{I} + \tau = 0 \\ &\partial\Omega \equiv \partial\Omega_1 \cup \partial\Omega_2 \end{aligned} \tag{5.14}$$

where $\partial\Omega$ is the boundary of the entire domain Ω.

The boundary conditions for the pressure correction given by equation (5.11) are the following:

$$\begin{aligned} &\text{on } \partial\Omega_1 && \frac{\partial\phi}{\partial n} = 0 \\ &\text{on } \partial\Omega_2 && \phi = 0 \end{aligned} \tag{5.15}$$

5.2.3 Finite-Volume Discretization

The equations are discretized using the finite-volume approach with staggered grid

where velocity components and temperature are defined at cell vertices (corners) and pressure and pressure correction variable are defined at cell centers in order to avoid oscillation in pressure field due to decoupling (Fig.5.1).

A compact scheme is described with all operations being restricted to within each grid-cell, which makes the algorithm suitable for use of adaptive unstructured (locally refined and triangular) grids. Artificial dissipation terms are added to both the momentum and energy equations.

The normalized system of the three-dimensional incompressible Navier-Stokes equations are given in integral form for a bounded domain V as follows:

$$\frac{d}{dt}\iiint_V \mathbf{U}\, dV + \iiint_V \left(\frac{\partial \mathbf{F}}{\partial x} + \frac{\partial \mathbf{G}}{\partial y} + \frac{\partial \mathbf{H}}{\partial z}\right) dV = 0 \tag{5.16}$$

where **U** is the state vector and **F**, **G** and **H** are the convective and viscous flux vectors in the x, t, and z- directions respectively.

Consider the typical prismatic grid-cell in Fig.5.1 which has two triangular and three quadrilateral faces. The volume integral containing the spatial derivatives in equation (5.16) is equivalent to a surface integral via the divergence theorem:

$$\iiint_V \left(\frac{\partial \mathbf{F}}{\partial x} + \frac{\partial \mathbf{G}}{\partial y} + \frac{\partial \mathbf{H}}{\partial z}\right) dV = \int_{\partial V} \left(\mathbf{F}\hat{i} + \mathbf{G}\hat{j} + \mathbf{H}\hat{k}\right) \hat{n}\, dS \tag{5.17}$$

where $\hat{i}, \hat{j}, \hat{k}$ are the unit vectors in the x, y, z- directions, and $\hat{n}$ is the unit vector normal to the cell surface ∂V. This surface integral is discretized as:

$$\sum_{f=1}^{n_f} \left[\mathbf{F}S_x + \mathbf{G}S_y + \mathbf{H}S_z\right]_f \tag{5.18}$$

where the sumation is over the faces of prisms (n_f=5) and S_x, S_y, S_z are the face-areas projected on the yz, xz and xy-planes, respectively. The flux vectors are considered at the face centers and their values are obtained by averaging the values at the face-nodes or cell-centers.

The viscous terms contain higher order spatial derivatives than the inviscid terms of the Navier-Stokes equations. As a consequence, the required numerical molecule for their treatment is usually larger. In the present work, a compact scheme is considered

that has the same numerical molecule as the inviscid terms. Stress terms are computed at center of each cell using information of the velocity component on corner nodes, then effect of viscous terms to corner nodes are computed using *dual* cells. Pressure terms in Navier-Stokes equations are computed because pressure field is defined at cell center in this formulation.

5.2.4 Artificial Dissipation

For high Reynolds number flows, the nonlinear convection terms are dominant. Central space differencing schemes are susceptible to oscillatory solution modes. In the present work, a fourth-order difference smoothing term is added explicitly to the momentum and energy transport equations in order to suppress odd-even decoupling of the solution [34,38,96].

The smoothing operator is cast in a form suitable for adaptive unstructured grids. All operations are split in such a way that no information is required from outside of each cell. Each grid node receives contributions from each one of its surrounding cells. The operator is formed in two steps. The second order difference operator is formed in the first step. In Fig.5.2, There are 6 triangular faces named A, B, C, D, E, F and 7 corner nodes, 0, 1, 2, 3, 4, 5, 6. Two layers of prisms are generated and prism cells in the first layer are A1, B1, C1, D1, E1, F1 and those in the second layers are A2, B2, C2, D2, E2, F2. Nodes in the top surface of first layer are 10, 11, 12, 13, 14, 15, 16 and those in the second layer are 20, 21, 22, 23, 24, 25, 26. The second order distributions for variable ϕ (u, v, w and T) to cell-corner 10 in Fig.5.2 are as follows.

$$D^2_{A1-10}(\phi) = \phi_0 + \phi_1 + \phi_2 + \phi_{10} + \phi_{11} + \phi_{12} - 6\phi_{10}$$

$$D^2_{B1-10}(\phi) = \phi_0 + \phi_2 + \phi_3 + \phi_{10} + \phi_{12} + \phi_{13} - 6\phi_{10}$$

$$D^2_{C1-10}(\phi) = \phi_0 + \phi_3 + \phi_4 + \phi_{10} + \phi_{13} + \phi_{14} - 6\phi_{10}$$

$$D^2_{D1-10}(\phi) = \phi_0 + \phi_4 + \phi_5 + \phi_{10} + \phi_{14} + \phi_{15} - 6\phi_{10}$$

$$D^2_{E1-10}(\phi) = \phi_0 + \phi_5 + \phi_6 + \phi_{10} + \phi_{15} + \phi_{16} - 6\phi_{10}$$

$$D^2_{F1-10}(\phi) = \phi_0 + \phi_6 + \phi_1 + \phi_{10} + \phi_{16} + \phi_{11} - 6\phi_{10}$$

$$D^2_{A2-10}(\phi) = \phi_{10} + \phi_{11} + \phi_{12} + \phi_{20} + \phi_{21} + \phi_{22} - 6\phi_{10}$$

$$D^2_{B2-10}(\phi) = \phi_{10} + \phi_{12} + \phi_{13} + \phi_{20} + \phi_{22} + \phi_{23} - 6\phi_{10}$$

$$D^2_{C2-10}(\phi) = \phi_{10} + \phi_{13} + \phi_{14} + \phi_{20} + \phi_{23} + \phi_{24} - 6\phi_{10}$$

$$D^2_{D2-10}(\phi) = \phi_{10} + \phi_{14} + \phi_{15} + \phi_{20} + \phi_{24} + \phi_{25} - 6\phi_{10}$$

$$D^2_{E2-10}(\phi) = \phi_{10} + \phi_{15} + \phi_{16} + \phi_{20} + \phi_{25} + \phi_{26} - 6\phi_{10}$$

$$D^2_{F2-10}(\phi) = \phi_{10} + \phi_{16} + \phi_{11} + \phi_{20} + \phi_{26} + \phi_{21} - 6\phi_{10} \qquad (5.19)$$

The second step duplicates the first, replacing state variables by second order differences from the first step. The fourth order smoothing distributions are:

$$D^4_{A1-10}(\phi) = D^2_0 + D^2_1 + D^2_2 + D^2_{10} + D^2_{11} + D^2_{12} - 6D^2_{10}$$

$$D^4_{B1-10}(\phi) = D^2_0 + D^2_2 + D^2_3 + D^2_{10} + D^2_{12} + D^2_{13} - 6D^2_{10}$$

$$D^4_{C1-10}(\phi) = D^2_0 + D^2_3 + D^2_4 + D^2_{10} + D^2_{13} + D^2_{14} - 6D^2_{10}$$

$$D^4_{D1-10}(\phi) = D^2_0 + D^2_4 + D^2_5 + D^2_{10} + D^2_{14} + D^2_{15} - 6D^2_{10}$$

$$D^4_{E1-10}(\phi) = D^2_0 + D^2_5 + D^2_6 + D^2_{10} + D^2_{15} + D^2_{16} - 6D^2_{10}$$

$$D^4_{F1-10}(\phi) = D^2_0 + D^2_6 + D^2_2 + D^2_{10} + D^2_{16} + D^2_{11} - 6D^2_{10}$$

$$D^4_{A2-10}(\phi) = D^2_{10} + D^2_{11} + D^2_{12} + D^2_{20} + D^2_{21} + D^2_{22} - 6D^2_{10}$$

$$D^4_{B2-10}(\phi) = D^2_{10} + D^2_{12} + D^2_{13} + D^2_{20} + D^2_{23} + D^2_{23} - 6D^2_{10}$$

$$D^4_{C2-10}(\phi) = D^2_{10} + D^2_{13} + D^2_{14} + D^2_{20} + D^2_{23} + D^2_{24} - 6D^2_{10}$$

$$D^4_{D2-10}(\phi) = D^2_{10} + D^2_{14} + D^2_{15} + D^2_{20} + D^2_{24} + D^2_{25} - 6D^2_{10}$$

$$D^4_{E2-10}(\phi) = D^2_{10} + D^2_{15} + D^2_{16} + D^2_{20} + D^2_{25} + D^2_{26} - 6D^2_{10}$$

$$D^4_{F2-10}(\phi) = D^2_{10} + D^2_{16} + D^2_{11} + D^2_{20} + D^2_{26} + D^2_{21} - 6D^2_{10} \tag{5.20}$$

The values of u', v' and w' are updated as follows:

$$\begin{aligned} u'_j &= u^{(n)}_j + \delta u_j \\ v'_j &= v^{(n)}_j + \delta v_j \\ w'_j &= w^{(n)}_j + \delta w_j \end{aligned} \tag{5.21}$$

The fourth-order difference terms of u, v and w are added to stabilize the solution as follows:

$$\begin{aligned} u'_j &= u^{(n)}_j + \delta u_j + \sigma_4(u) D^4_j(u^{(n)}_j) \\ v'_j &= v^{(n)}_j + \delta v_j + \sigma_4(v) D^4_j(v^{(n)}_j) \\ w'_j &= w^{(n)}_j + \delta w_j + \sigma_4(w) D^4_j(w^{(n)}_j) \end{aligned} \tag{5.22}$$

where σ_4 is an empirical coefficient. Criterion for the determination σ_4 strictly depends on grid size and Reynolds number. For higher *Re* number flow and coarser grid, σ_4 should be larger. Large value of σ_4 stabilizes the solution but sometimes destroys the accuracy. Therefore, special care is required for choosing the value of σ_4.

5.2.5 Time-Step Calculation

Using central space and forward time differencing, the stability limitation for the model 1-D convection equation:

$$u_t + cu_x = 0 \tag{5.23}$$

is limited by CFL limitation:

$$\frac{c\Delta t}{\Delta x} \le 1 \tag{5.24}$$

The corresponding stability restriction for the 1-D model diffusion equation:

$$u_t + \nu u_{xx} = 0 \tag{5.25}$$

is:

$$\frac{\nu\Delta t}{\Delta x^2} \le \frac{1}{2} \tag{5.26}$$

Comparing the two time step limitations, we obtain:

$$\frac{\Delta t_{vis}}{\Delta t_{inv}} = \frac{1}{2}\frac{c\Delta x}{\nu} \approx \frac{\Delta x}{Re} \tag{5.27}$$

Therefore it is apparent that in most common cases the CFL stability restriction is much more severe than the viscous time step limitation. Only in cases of low Re, the viscous limitation can be severe. In the present scheme, a combination of the two limitations is employed. Specifically,

$$\Delta t = \omega \min\left\{ \frac{\Delta l}{|u| + \frac{\nu}{\alpha\Delta l}}, \frac{\Delta m}{|v| + \frac{\nu}{\alpha\Delta m}}, \frac{\Delta n}{|w| + \frac{\nu}{\alpha\Delta n}} \right\} \tag{5.28}$$

where $\Delta l, \Delta m, \Delta n$ are the cell dimensions in the l, m, n cell directions, u, v, w are the corresponding velocity components, ν is the kinematic viscosity coefficient, and $\alpha = 1/2$ is the diffusion stability limitation. Lastly, ω is a safety factor which is usually less than 1.00.

5.2.6 Poisson Equation Treatment

After solving the momentum equations, the pressure and velocity fields should be corrected so that they satisfy the continuity equation. This process is performed by solving the Poisson equation for pressure correction. The Poisson equation (5.11) in

three-dimensional space can be written as follows:

$$\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + \frac{\partial^2\phi}{\partial z^2} = -\left(\frac{\partial u'}{\partial x} + \frac{\partial v'}{\partial y} + \frac{\partial w'}{\partial z}\right) \tag{5.29}$$

The velocity and pressure fields are corrected using equations (5.9), (5.10) and (5.29).

$$\begin{aligned} u^{(n+1)} &= u' - \frac{\partial\phi}{\partial x} \\ v^{(n+1)} &= v' - \frac{\partial\phi}{\partial y} \\ w^{(n+1)} &= w' - \frac{\partial\phi}{\partial z} \\ p^{(n+1)} &= p^{(n)} + \frac{1}{\Delta t}\phi \end{aligned} \tag{5.30}$$

The values for ϕ on the left hand side of (5.30) are treated implicitly, which requires solution of a system. The matrix that requires inversion is a symmetric positive definite matrix. The system is solved by the preconditioned CG (Conjugate Gradient) iterative method.

5.3 Multigrid Method

5.3.1 Multigrid Procedure

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 method, as described in [107]. If we have obtained the following linear system on a fine grid :

$$A_F\, u_F = f \tag{5.31}$$

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

$$u_F^{(i+1)} = u_F^{(i)} + R^T\, A_C^{-1}\, R\, (f - A_F\, u_F^{(i)}) \tag{5.32}$$

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 :

I. 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*.
II. Calculate the residual term on the fine grid by $r_F = f - A_F\, u_F^{(i)}$.
III. *Restric*t the residual term on to the coarse grid by $r_C = R\, r_F$.
IV. 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 [11,107,128].
V. Interpolate (or *prolong*) the coarse grid correction on the fine grid by $\Delta u_C^{(i)} = R^T\, u_C$.
VI. 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 procedure to consecutive systems of

coarse-grid equations gives a multigrid V-cycle [10,11,110] (Fig.5.3 and Fig.5.4). 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 [107]. Alternatively, in a W-cycle [10,11,110] (Fig.5.3), 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.

In [48], various types of prolongation and restriction methods are compared on 2D structured grids.

5.3.2 Multigrid as a Preconditioiner

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 [87,121].

Mutigrid solvers and Krylov iterative solvers preconditioned by multigrid are intrinsically suitable for parallel computing. In [4], MGCG and multigrid (MG) are compared under various conditions for groundwater flow simulation using a CRAY-T3D computer with 256 PEs. MG itself is competent with MGCG for small problems, but slows down in large cases with more PEs. MG diverges for strongly heterogeneous problems, while MGCG remains robust.

5.3.3 Semi-Coarsening

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 [6,12,52,53,54,58,64,66,95,117].

5.3.4 Geometric and Algebraic Multigrid

One of the most important issues in multigrid is the construction of the coarse grids. There are 2 basic multigrid approaches; geometric [1,2,23,90] and algebraic [16,58,59,60,109,110,112]. 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. Algebraic multigrid method is suitable for applications with unstructured grids. Many tools for both geometric and algebraic methods on unstructured grids have been developed [1,2,3,67].

Multigrid methods for locally refined grids have been also described in [17,89,104,116,122]. They are mainly for block-structured grids and very few methods for unstructured grids were developed [90].

5.3.5 Other Recent Studies in Multigrid

Recently, many examples have been presented for very large scale problems on massively parallel computers, as shown in [1,2,3,4,5,23,60,61,62].

Multigrid methods were originally developed for solving linear equations [10,11], then applied to coupled linearlized equations, as shown in [49]. In recent years, multigrid method has been applied to solving nonlinear systems. In Full Approximation Scheme (FAS) [10,11], nonlinear Gauss-Seidel smoothing with multigrid is used for solving nonlinear equations. In Newton-Klyrov method [47] for nonlinear applications, inexact Newton nonlinear iterative method is applied using GMRES [7] solver with multrigrid preconditioning.

5.4 Parallel Multigrid Preconditioned Iterative Solvers

5.4.1 Problem Definitions

In this work, the target application is 3D incompressible thermal convection in the region between two 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 described in 5.2 is applied, in which momentum and energy equations are solved explicitly, and the pressure-correction Poisson equation (5.11) 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.5.5). Meshes are initiated from an icosahedron and are globally refined recursively as in Fig.5.6. The grid hierarchy defined by recursive refinement can be utilized as coarse grid formation (Fig.5.7 and Fig.5.8). 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, as shown in Fig. 5.1.

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.

5.4.2 Parallel MGCG Solvers for Poisson Equations

(1) Basic Procedure

The parallel MGCG solver was implemented in Fortran 90 using flat MPI parallel programming model. The features of the procedure are summarized as follows [74,78,80]:

（1） V-cycle MGCG solver.
（2） Gauss-Seidel or ILU(0) smoothing.
（3） Semi-coarsening in lateral and normal-to-surface direction (Fig.5.9).
（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 [7] was also tested as a smoother for the preconditioning component of the CG iterative method. ILU smoothers represent a class of smoothing procedures all of which are based on an incomplete LU factorization of the matrix. A variety of such smoothing procedures has been proposed and investigated [110]. According to [110], certain versions of ILU type smoothing are regarded as particularly robust for anisotropic 2D problems. However, robust ILU smoothers for general anisotropic equations are more complicated and expensive for 3D problems.

In the current work, *localized* ILU(0) procedure described in Chpter 2 has been implemented recursively as a multi-level smoother [71,72,73,78,79,80,81,131].

The computational procedure for the preconditioned CG method for solving $Ax = b$ is as follows [7] :

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.5.3 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 shown in Fig.5.10. In this procedure, direct injection method described in [48] is implemented for restriction procedure due to its simplicity. The ILU(0) smoother has been implemented with the additive Schwartz domain decomposition method [107] 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 this work, semi-coarsening in normal-to-surface directions are done successively, then operations in lateral directions are done after that. 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 (Fig.5.11). The equation on the coarsest grid (20×2=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.

(2) Data Structure

Extended local data structure based on that of GeoFEM has developed for multilevel parallel procedure was developed. 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 [83,131]. In this work, 2 types of communication tables are defined, as shown in Fig.5.12 and Fig.5.13. Staggered cell was adopted in this work where velocity components and temperature are defined at cell corners and pressure and pressure correction potential are defined at cell centers. Two types of communication tables are for *node-based* and *cell-based* variables respectively. Node-based variables appear only in the finest grids. But as for the cell-based variables, multilevel communication tables were defined at each level according to the multigrid procedure (Fig.5.14) [74,78,80].

Node-based variables are computed through only explicit procedure, such as explicit time-marching scheme for momentum and energy equations and update of velocity field by pressure correction. In contrast, cell-centered variables require implicit computation such as solving linear equations. Therefore, 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 is very similar to that of interface nodes among processors in GeoFEM. This type of communication table is defined at each grid level of refinement [74,78,80].

(3) Tested Patterns

Several patterns have been considered for the preconditioners and the combination of smoothers on parallel and serial multigrid procedures as follows (Fig.5.15) [74,78,80] :

- ICCG
- ICCG/ASDD (ICCG with Additive Schwartz Domain Decomposition)
- MGCG/FGS (Full Gauss-Seidel)

- MGCG/GSp (Gauss-Seidel-Parallel)
- MGCG/ILU-GSp (ILU-Gauss-Seidel-Parallel)
- MGCG/ILU-GSs (ILU-Gauss-Seidel-Serial)

In *MGCG/FGS*, multi-level Gauss-Seidel smoother was applied to both parallel and serial procedures for multigrid procedure in Fig.5.11. Effect of parameters for restriction and prolongation were evaluated.

In *MGCG/GSp*, multi-stage Gauss-Seidel smoother was applied to the parallel procedure recursively but single-stage parallel Gauss-Seidel smoothing was applied to the coarsest level of the grid instead of serial procedure. Effect of iteration number in the parallel Gauss-Seidel procedure is a very important parameter, because the accuracy of the solution on the coarse grid affects the convergence of the entire multigrid system [11,107,128] but many Gauss-Seidel iterations require frequent communications. Communication cost for this parallel Gauss-Seidel procedure is very expensive, because only 20 cells are included in one PE/

In *MGCG/ILU-GSp*, multi-stage ILU(0) smoothing is applied to parallel multigrid procedure and single-stage parallel Gauss-Seidel smoothing was applied instead of serial procedure. Effect of iteration number in the parallel Gauss-Seidel procedure is a very important parameter as is in *MGCG/GSp*. In both MGCG/GSp and MGCG/ILU-GSp, problem size for parallel Gaus-Seidel procedure, which depends on total PE number, is 20 × PE#. Therefore parallel performance can be worse for the cases with large PE number.

Finally, in *MGCG/ILU-GSs*, parallel Gauss-Seidel is applied to the serial procedure as in *MGCG/ILU-GSp*.

5.4.3 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 [76,89,91].

Here, a dynamic adaptation algorithm developed by the author for 3D unstructured meshes [76,89,91] 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 [76,89,91]. 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.5.16 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 (Fig.5.17) [74,78,80,89,91].

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 [11] 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.5.18). 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. 5.18.

5.5 Examples (Poisson Equations)

5.5.1 Outline

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 [132,134] with up to 128 processors. Here, 2 problems were considered. In both problems the following homogeneous Poisson equation was solved :

$$\Delta\phi = 1$$

In this subsection, two types of meshes were considered. One is uniform surface grid (Poisson-I) and the other is locally refined surface meshes (Poisson-II).

5.5.2 Poisson-I (Uniform Mesh)

In the 1st application (Poisson-I), the problem size for 1 processor was fixed at 320 triangles (level=2 in Fig.5.6) × 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 5.19 and 5.20 compare the results (elapsed computation time including communication and parallel performance) for ICCG, MGCG/FGS (Full-Gauss-Seidel) and MGCG/ILU-GSp (ILU-Gauss-Seidel-Parallel) computations in Poisson-I for both of uniform and single-patch boundary conditions. Parallel performance was computed from the ratio of purely parallel computing time except communication overhead and serial processes for solving linear equations and entire solver time. The computation

time for MGCG/FGS and MFCG/ILU-GSp 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 and ILU(0) smoothing [71,72,73,79,81], approaching Jacobi smoothing as the number of PEs increases. However, even in the 128 PE cases, MGCG/FGS and MFCG/ILU-GSp are much faster than ICCG. MGCG/ILU-GSp exhibits more robust convergence than MGCG/FGS, particularly for the cases with the *single-patch* boundary condition. Parallel performance of ICCG and MGCG/FGS is almost 100% but that of MGCG/ILU-GS decreases as PE number increases. Performance is about 90% with 128 PEs. This is due to the communication overhead for parallel Gauss-Seidel procedure at the coarsest level of the grid. Fig.5.21 shows comparison between ICCG and MGCG/FGS up to 16 PEs with single-patch boundary condition. MGCG/FGS provides nice scalability.

Figure 5.22 compares ICCG and ICCG/ASDD. Iteration number for convergence of ICCG/ASDD is as half as that of ICCG but computation time is almost competitive.

Figure 5.23 shows the number of multigrid cycle of MGCG/FGS for both uniform and single-patch boundary conditions. Number of multigrid cycle increase according to PE number, especially in the cases with single-patch boundary condition.

Figures 5.24 and 5.25 compares MGCG/FGS and MGCG/GSp (Gauss-Seidel-Parallel). These two methods are competitive from the view point of computation time but parallel performance of MGCG/GSp is going down to 95% in the cases with 128 PEs. MGCG/GSp does not converge in certain number of iterations with one-patch boundary condition, if the number of PE is more than 64 (Fig.5.25).

Figures 5.26 and 5.27 compares MGCG/ILU-GSp and MGCG/ILU-GSs (Gauss-Seidel-Serial). These two methods are competitive, if the number of PE is less than 64, but MGCG/ILU-GSp outperforms, if the number of PE is more than 64. Parallel performance of MGCG/ILU-GSs remains almost 100%.

Figure 5.28 shows effect of iteration number for parallel Gauss-Seidel process in MGCG/ILU-GSp with 64 PEs. In cases with uniform B.C., iteration number around 100 provides good performance, but more iterations are required for cases with one-patch B.C. This is because that equation with single-patch B.C. provides worse condition for solving and requires more iterations for convergence. Thus, parallel performance in MGCG/ILU-GS is worse for cases with one-patch B.C. (Fig. 5.19 and 5.20).

Furthermore, effect of *clustered* grid spacing in radial direction was also evaluated. In this case, grid spacing near the inner and outer walls are fixed as 0.01 and increases as follows:

$$\Delta r_1 = 0.01,\ \Delta r_2 = \alpha \Delta r_1,\ \Delta r_i = \alpha \Delta r_{i-1} = \alpha^{i-1} \Delta r_1,\quad \Delta r_i \le 0.10$$

$$\Delta r_n = 0.01,\ \Delta r_{n-1} = \alpha \Delta r_n,\ \Delta r_{n-i} = \alpha \Delta r_{n-i-1} = \alpha^{i-1} \Delta r_n,$$

$$\Delta r_{n-i} \le 0.10$$

Both *uniform* and *one-patch* boundary conditions are evaluated. Figures 5.29-5.32 compare the results for ICCG, MGCG/FGS, MGCG/GSp, MGCG/ILU-GSp and MGCG/ILU-GSs using clustered mesh size in radial direction. In uniform boundary condition, each method shows similar behavior, as shown in cases with uniformly spaced meshes in radial direction. In one-patch boundary conditions, only MGCG/ILU-GSp converges in reasonable computation time, if the number of PE is more than 64.

Figures 5.19-5.32 provides following remarks:

- MGCG/FGS is best for the cases up to 32 PEs
- for the cases with up to 128 PEs, MGCG/FGS is best in uniform B.C and MGCG/ILU-GSp in single-patch B.C. MGCG/FGS does not converge for clustered mesh size in radial direction with single-patch B.C. if the number of PE is more than 64.

5.5.3 Poisson-II (Locally Refined Mesh)

In the 2nd application (Poisson-II), 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.5.6) with 320 triangles. As shown in Fig.5.33, 3-level grid adaptation was applied. At each adaptation level, the number of triangular facets varies as follows (Fig.5.33):

- 1st level: 532 triangles
- 2nd level: 1,508 triangles
- 3rd level: 4,448 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/FGS using between 2 and 32 PEs.

Figure 5.34 shows a comparison between the *direct jump* and *level-by-level* multigrid strategies for an adapted grid under the *uniform* boundary conditions with MGCG/FGS method. Performance was evaluated by computation time normalized by the number of cells (*i.e.*, degrees of freedom) on each processor according to problem size (*i.e.*, PE number). 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.

Figures 5.35, 5.36 and 5.37 show performance by ICCG, ICCG/ASDD and MGCG/FGS. MGCG/FGS provides much more *scalable* results than ICCG and ICCG/ASDD although performance is worse in the cases with 3-level adapted meshes in Fig.5.33. Performance for globally fine meshes and locally refined meshes are almost equal for a wide range of problem size.

Figures 5.38-5.43 compare results between the following two meshes:

- 3-level adapted mesh with 4,448 triangles
- globally-fine 4-level mesh with 5,120 triangles

under both uniform and single-patch boundary conditions. In cases with uniform boundary conditions, MGCG/FGS and MGCG/GSp are the best but MGCG/GSp provides better performance if the PE number is larger (Fig.5.39). In cases with single-patch boundary conditions, MGCG/FGS provides best performance.

5.6 Examples (Navier-Stokes Equations)

Finally, developed multigrid procedure has been applied to 3D Navier-Stokes equations with thermal convection in the region between two spherical surfaces. Very simple boundary conditions were applied as follows:

- $r = r_{min}$ (= 0.50): u = v = w = 0, T= T_{inn} (fixed).
- $r = r_{max}$ (= 1.00): u = v= w = 0, T= T_{out} (fixed).
- No heat generation
- *Uniform* boundary conditions for Poisson equations on $r = r_{max}$ surface.

Number of triangle surfaces is 5,120 (Fig.5.6), mesh size in radial direction is equally set to 1.25×10^{-3} and number of the layers in radial direction is 400. Therefore, total cell number is 2,048,000. In this computation, 16 PEs of the Hitachi SR2201 have been used. Each PE has 128,000 cells.

Table 5.1 shows elapsed computation time including communication overhead for the beginning 40 steps of the thermal-convection simulation. Most of the simulation process is spent for solving Poisson equations and updating velocity field. Therefore, performance of the linear solver is very critical.

In this case, MGCG/FGS and MGCG/ILU-GSp are much faster than ICCG. Between the two methods wish multigrid preconditioning, MGCG/FGS is slightly better.

5.7 Summary

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. Extended local data structure based on that of GeoFEM has been developed for the multilevel parallel procedure. Two types of communication tables, one for node-based variables and the other for cell-based variables, have been defined. Both Gauss-Seidel and ILU(0) with additive Schwartz domain decomposition smoothers have been tested. Various combinations of parallel and serial smoothers have been applied. The proposed procedure was applied to Poisson equations in the region between two spherical surfaces on adaptively generated semi-unstructured prismatic grids under various boundary conditions. Computational results obtained on a Hitachi SR2201 parallel computer using up to 128 processors demonstrate the good scalability of the method, as compared to ICCG solvers. Excellent parallel performance provided by the developed data structure is also demonstrated.

Among the tested methods, MGCG/FGS (Full-Gauss-Seidel) provides the best performance up to 32 PEs, while MGCG/ILU-GSp (ILU-Gauss-Seidel-Parallel, parallel Gauss-Seidel is applied for the coarsest level of the grid) is relatively robust for computations across many PEs, although parallel performance is worse for cases involving many PEs due to the communications overhead of the single-stage parallel Gauss-Seidel procedure. In the cases with clustered mesh spacing in the radial direction, MGCG/ILU-GSp provided more very convergence compared to other methods. Generally, ILU-type smoothers provide more robust convergence than Gauss-Seidel-type smoothers, especially for ill-conditioned problems.

The proposed procedure was also applied to grids with local refinement, and 2 multigrid strategies (*direct jump* and *level-by-level*) were compared. The *direct jump* method developed in this study was found to be much more efficient than the *level-by-level* method described in [11] for deeper-level adaptation despite the simplicity of the level-by-level method.

Finally, the proposed method was applied to 3D Navier-Stokes equations with thermal convection. CG solvers with multigrid preconditioning (MGCG/FGS and MGCG/ILU-GSp) provided much better performance than ICCG.

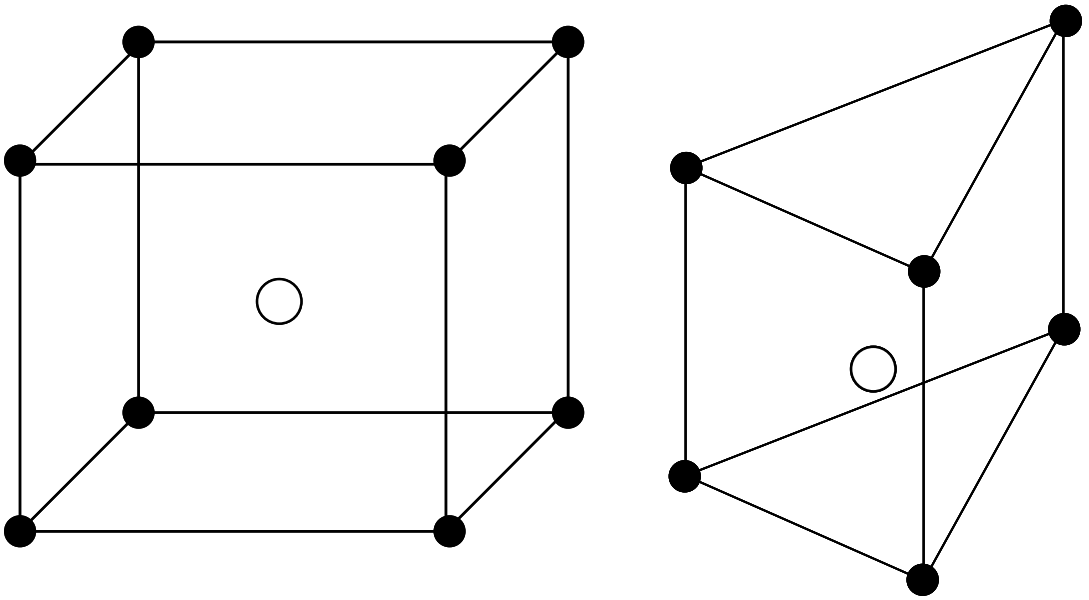


Fig. 5.1 Hexahedral and prismatic staggered cell: velocity components and temperature at corner nodes, pressure and pressure correction potential at cell center.

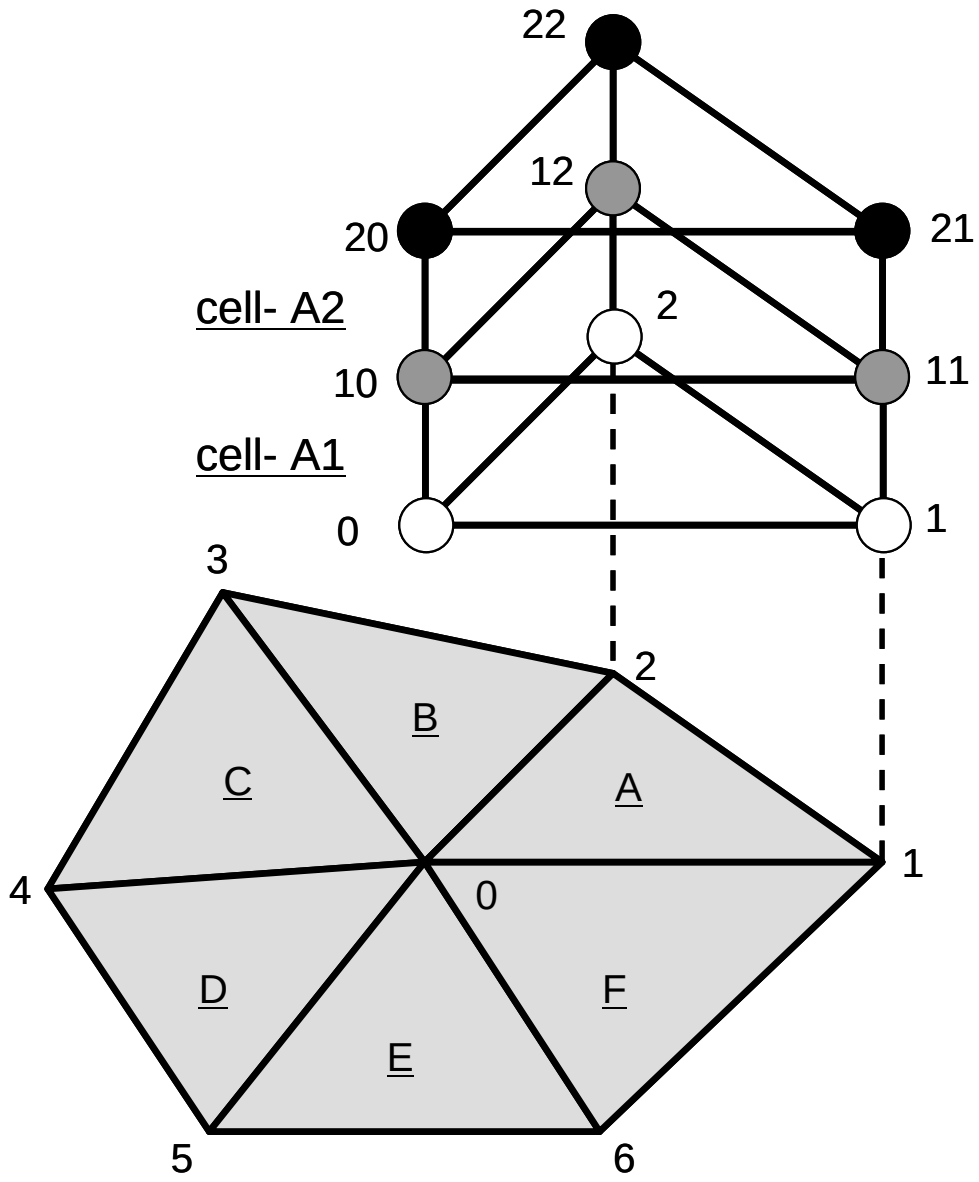


Fig. 5.2 2nd and 4th-order artificial dissipations for prismatic cells [42,74,76,78,80,89,91]

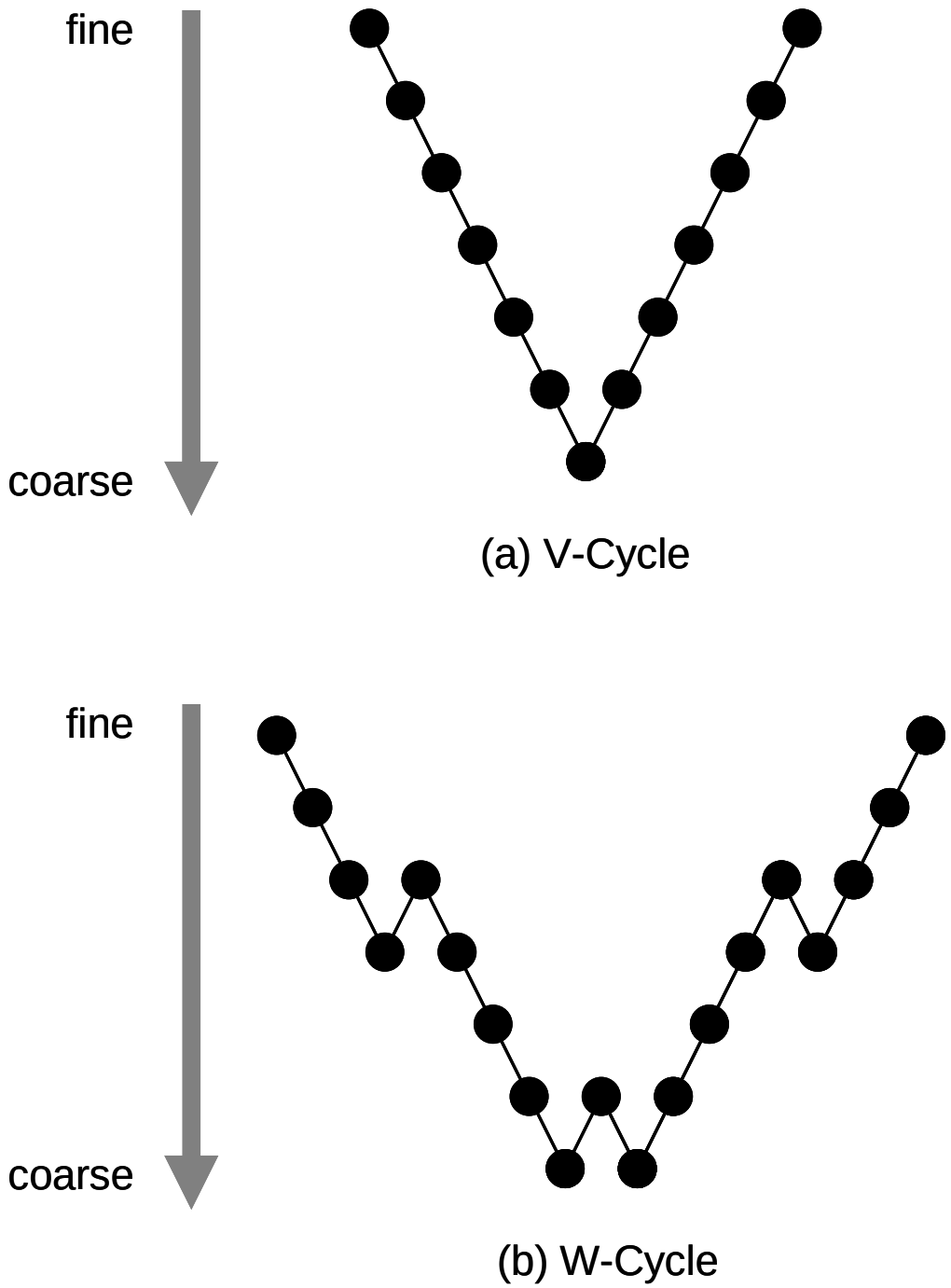
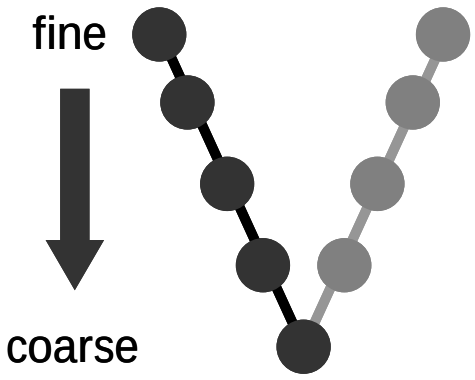


Fig. 5.3 V- and W-cycle for multigrid operation [11]



$L^k W^k = F^k$ **(Linear Equation: Fine Level)**

$R^k = F^k - L^k w_1^k$

$v^k = W^k - w_1^k$, $L^k v^k = R^k$

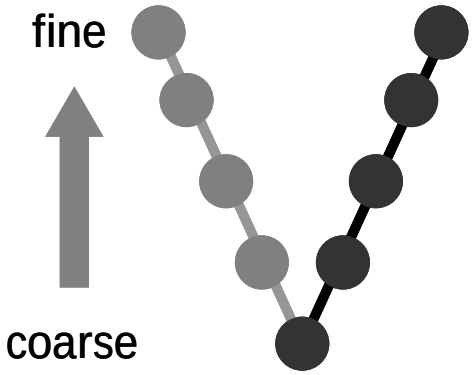
$R^{k-1} = I_k^{k-1} R^k$

$L^{k-1} v^{k-1} = R^{k-1}$ **(Linear Equation: Coarse Level)**

$v^k = I_{k-1}^k v^{k-1}$

$w_2^k = w_1^k + v^k$

w_1^k : Approx. Solution
v^k : Correction
I_k^{k-1} : Restriction Operator



$L^k W^k = F^k$ **(Linear Equation: Fine Level)**

$R^k = F^k - L^k w_1^k$

$v^k = W^k - w_1^k$, $L^k v^k = R^k$

$R^{k-1} = I_k^{k-1} R^k$

$L^{k-1} v^{k-1} = R^{k-1}$ **(Linear Equation: Coarse Level)**

$v^k = I_{k-1}^k v^{k-1}$

$w_2^k = w_1^k + v^k$

I_{k-1}^k : Prolongation Operator
w_2^k : Approx. Solution by Multigrid

Fig. 5.4 Detailed procedure of V-cycle multigrid operation [11]

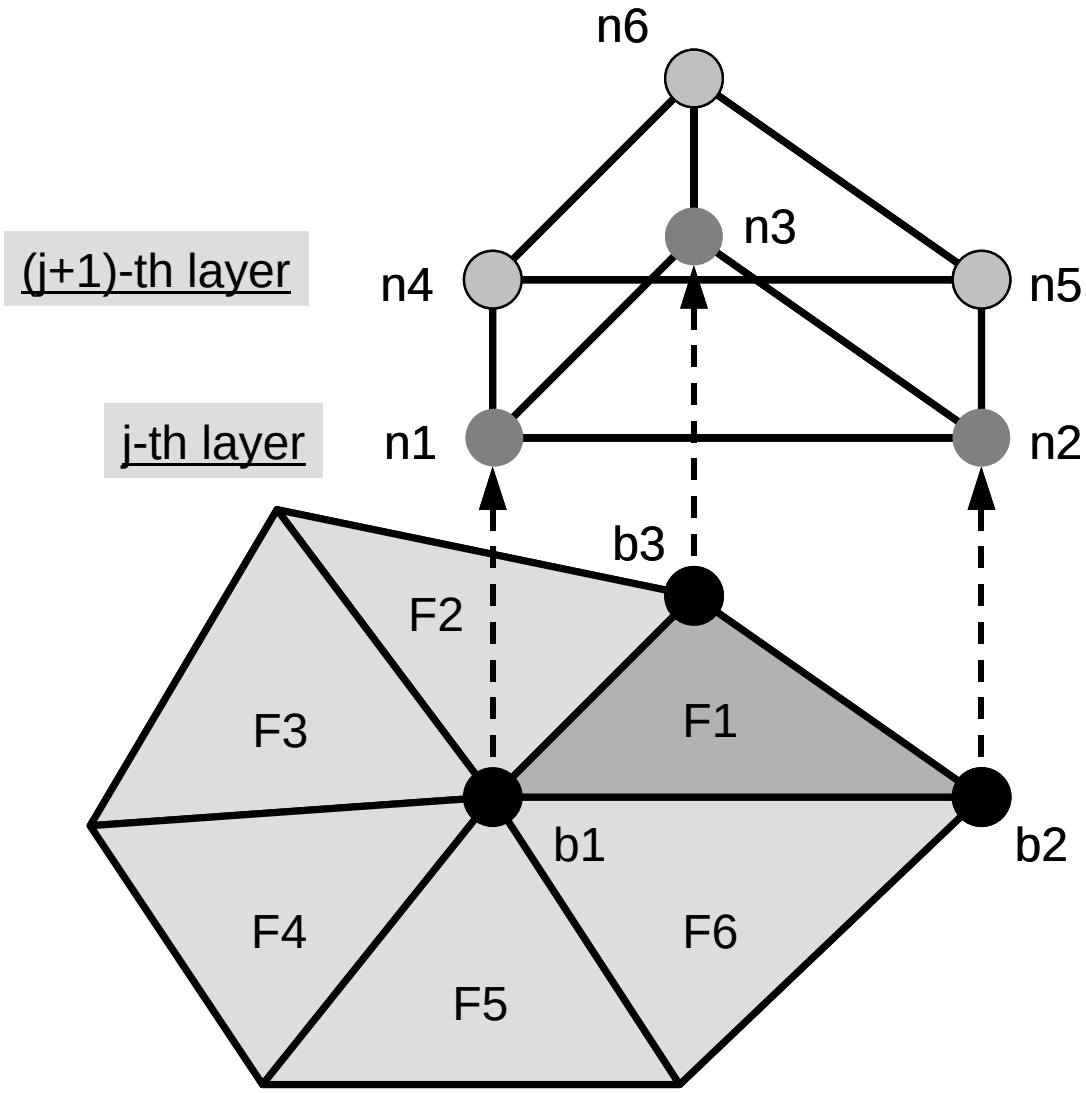


Fig. 5.5 Prisms generated from triangular facets

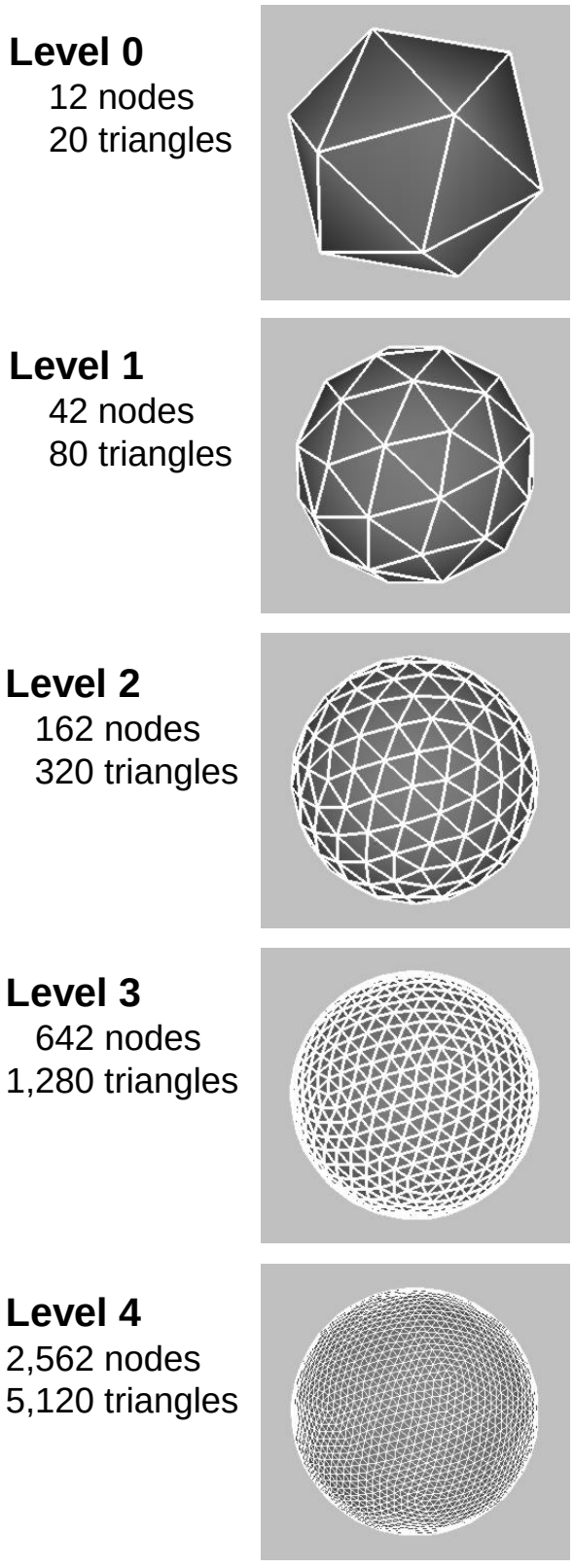


Fig. 5.6 Surface triangle meshes generated from icosahedron
4 children generated from 1 parent triangle [74,78,80]

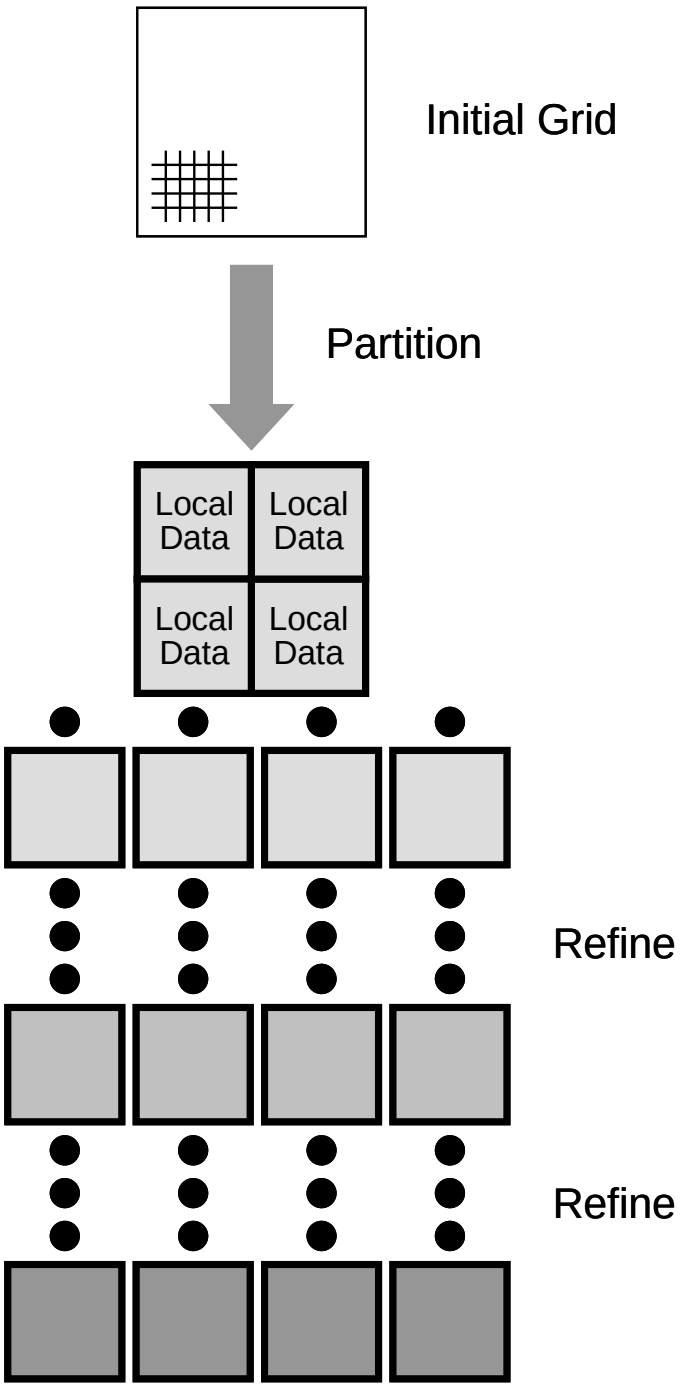


Fig. 5.7 Parallel grid generation in recursive manner [22,74,78,80]

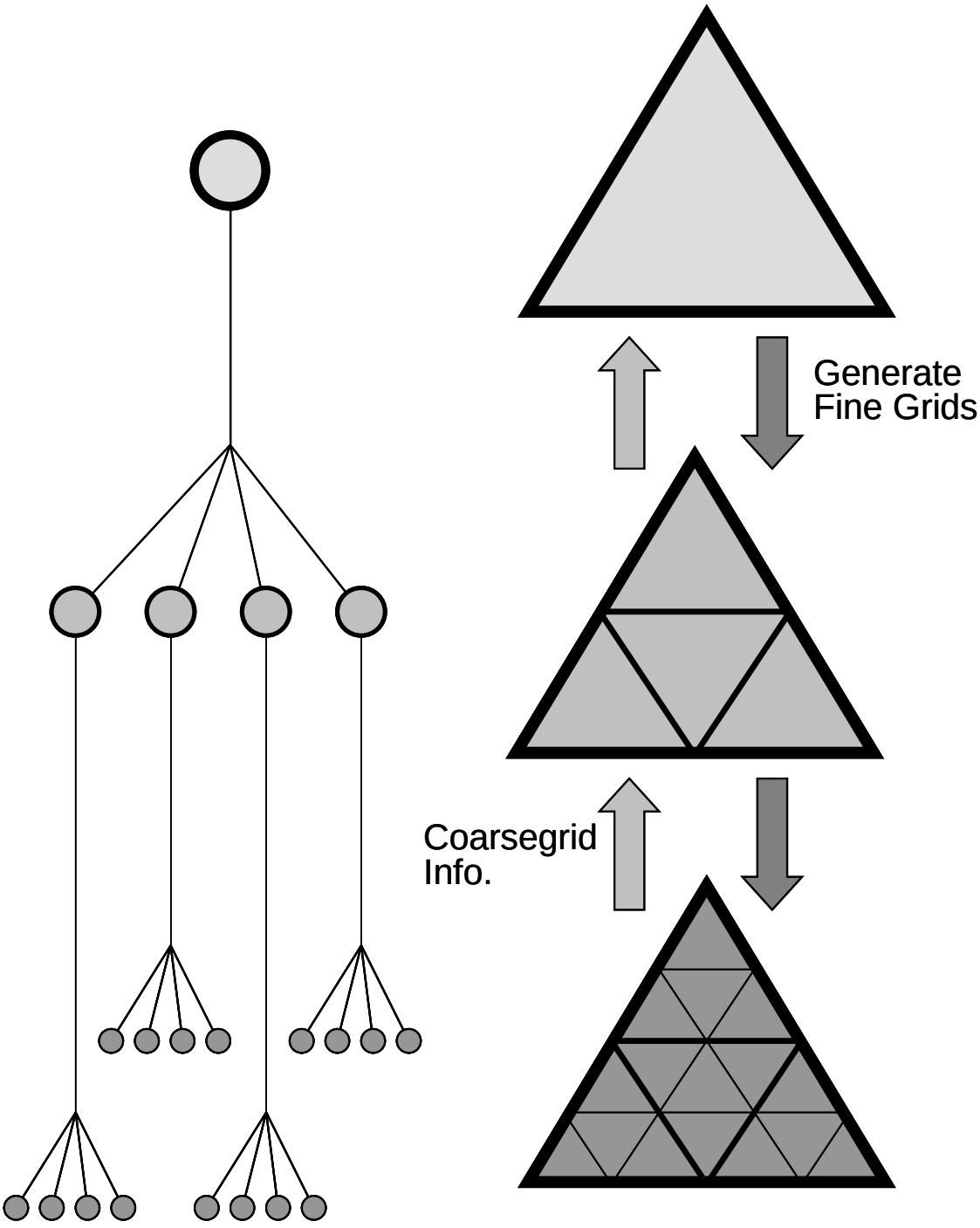


Fig. 5.8 Grid hierarchy from successive refinement [74,78,80]

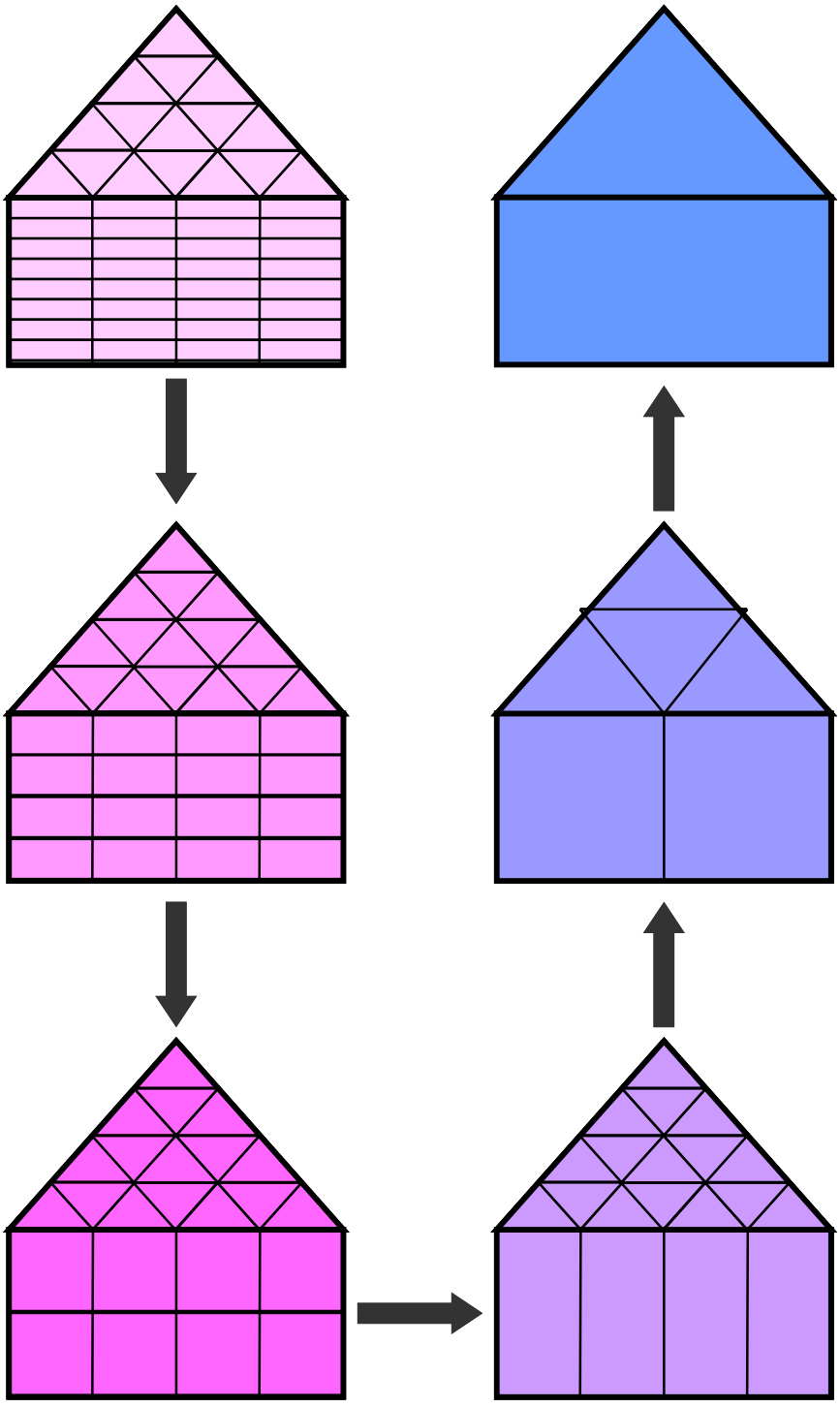
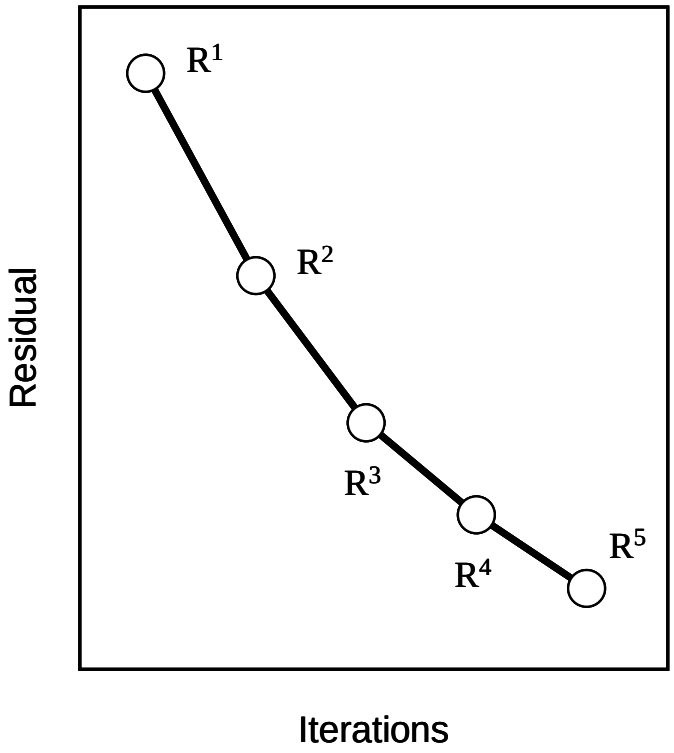


Fig. 5.9 Semi-coarsening in the lateral and normal-to-surface directions. In this work, semi-coarsening in normal-to-surface directions are done successively, then operations in lateral directions are done after that. [74,78,80]



if $R^i > \alpha R^{i-1}$ switch to the next stage of multigrid (α~0.80).

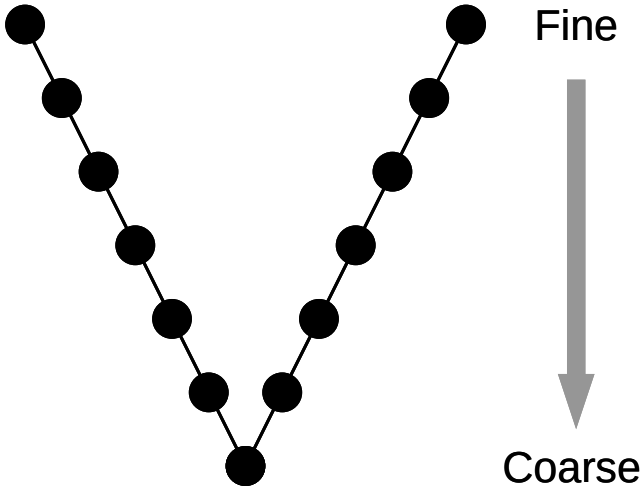


Fig. 5.10 Smoothing strategy at each restriction/prolongation stage [74,78,80]

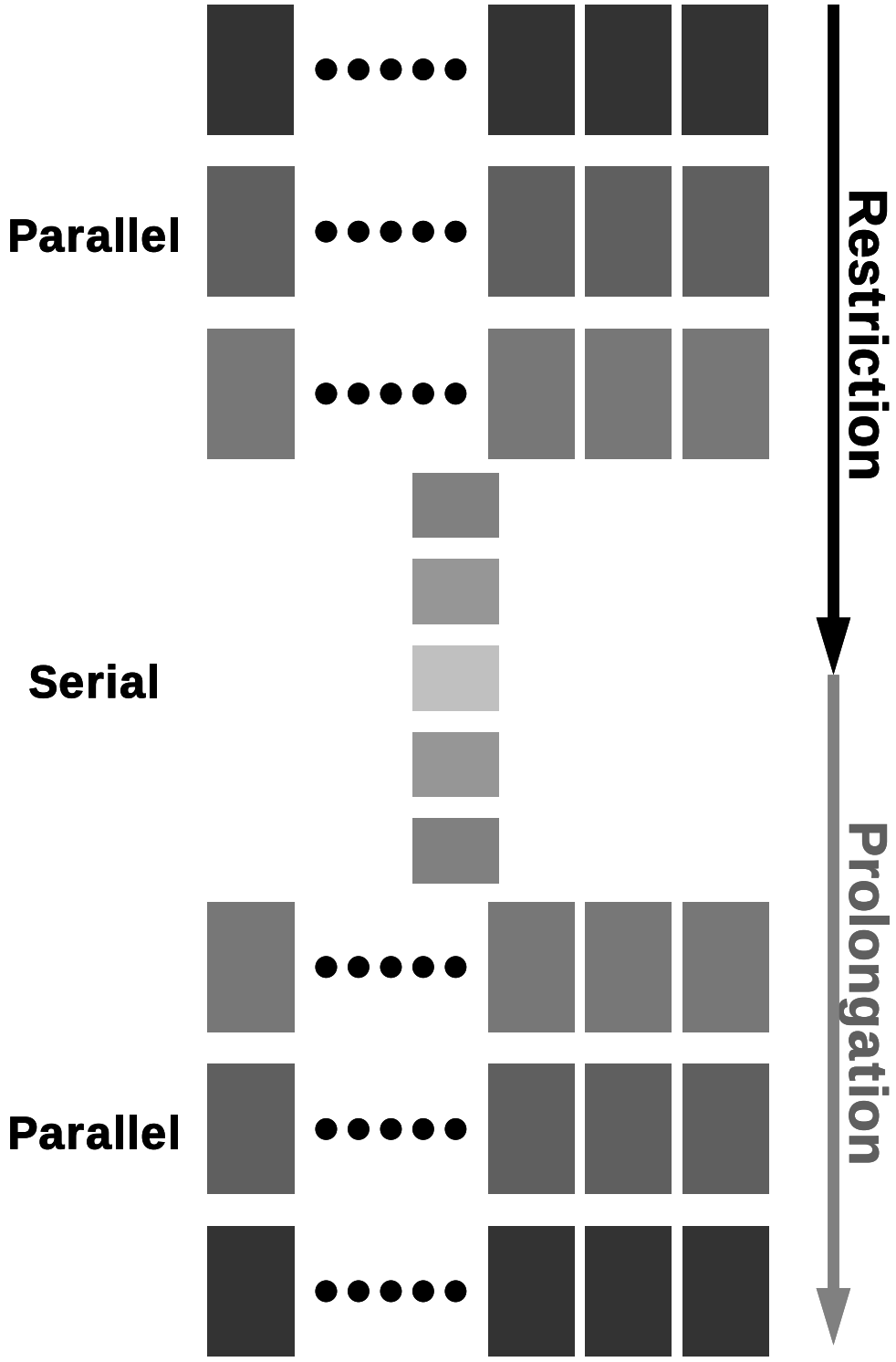


Fig. 5.11 Parallel and serial multigrid operations [74,78,80]

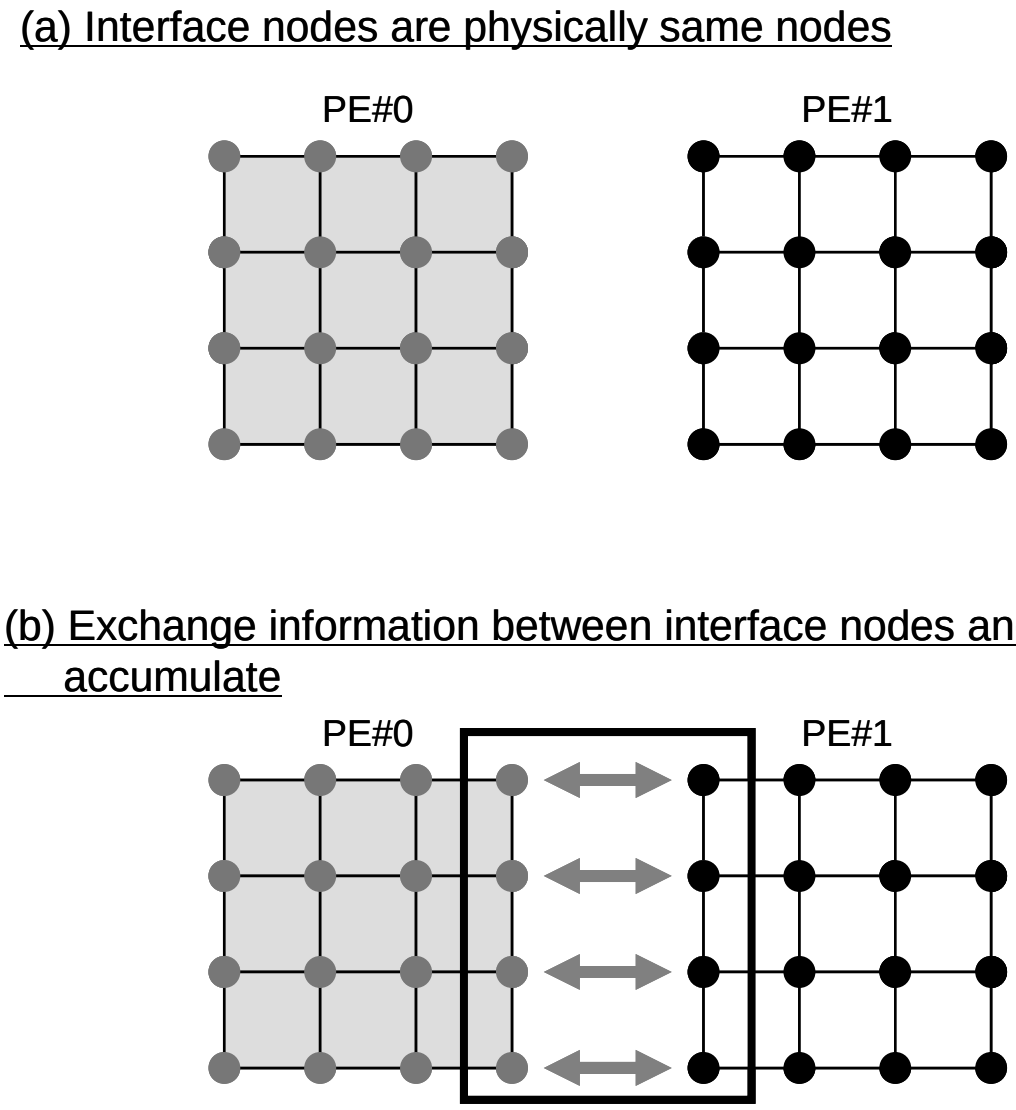
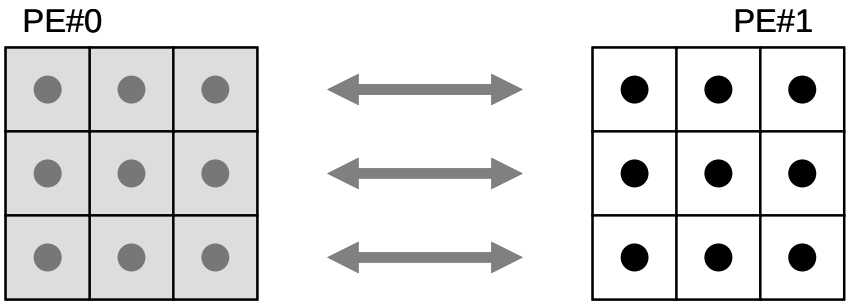


Fig. 5.12 Communication among domains (Momentum & Energy Equations) [74,78,80]

(a) requires EXTERNAL information for coefficient matrix formation



(b) Partition includes EXTERNAL elements

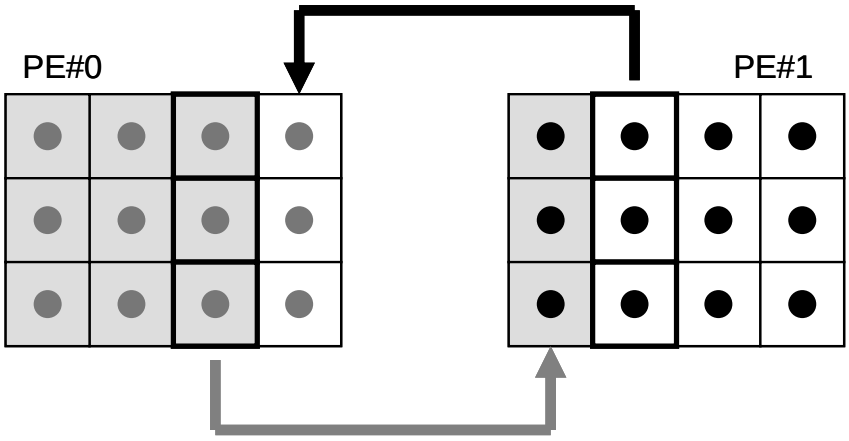


Fig. 5.13 Communication among domains (Poisson Equations) [74,78,80]

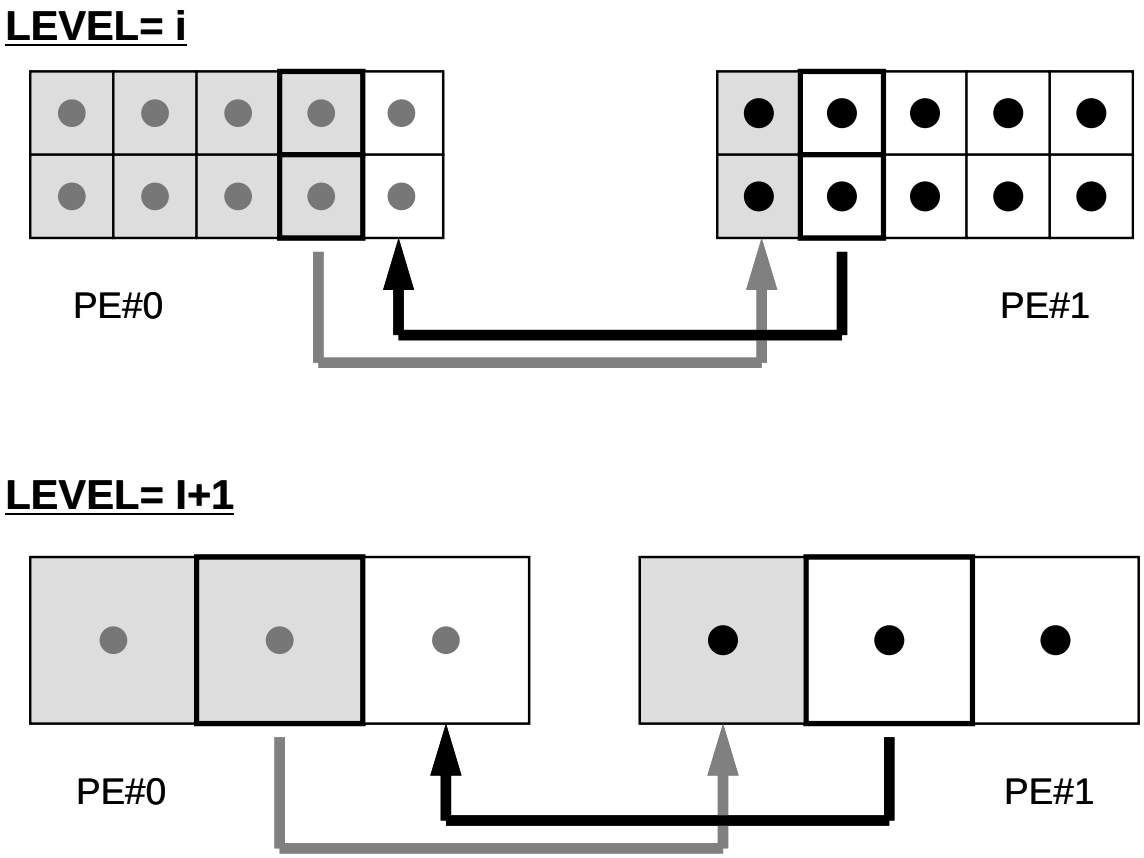


Fig. 5.14 Multilevel communication table (Poisson Equations) [74,78,80]

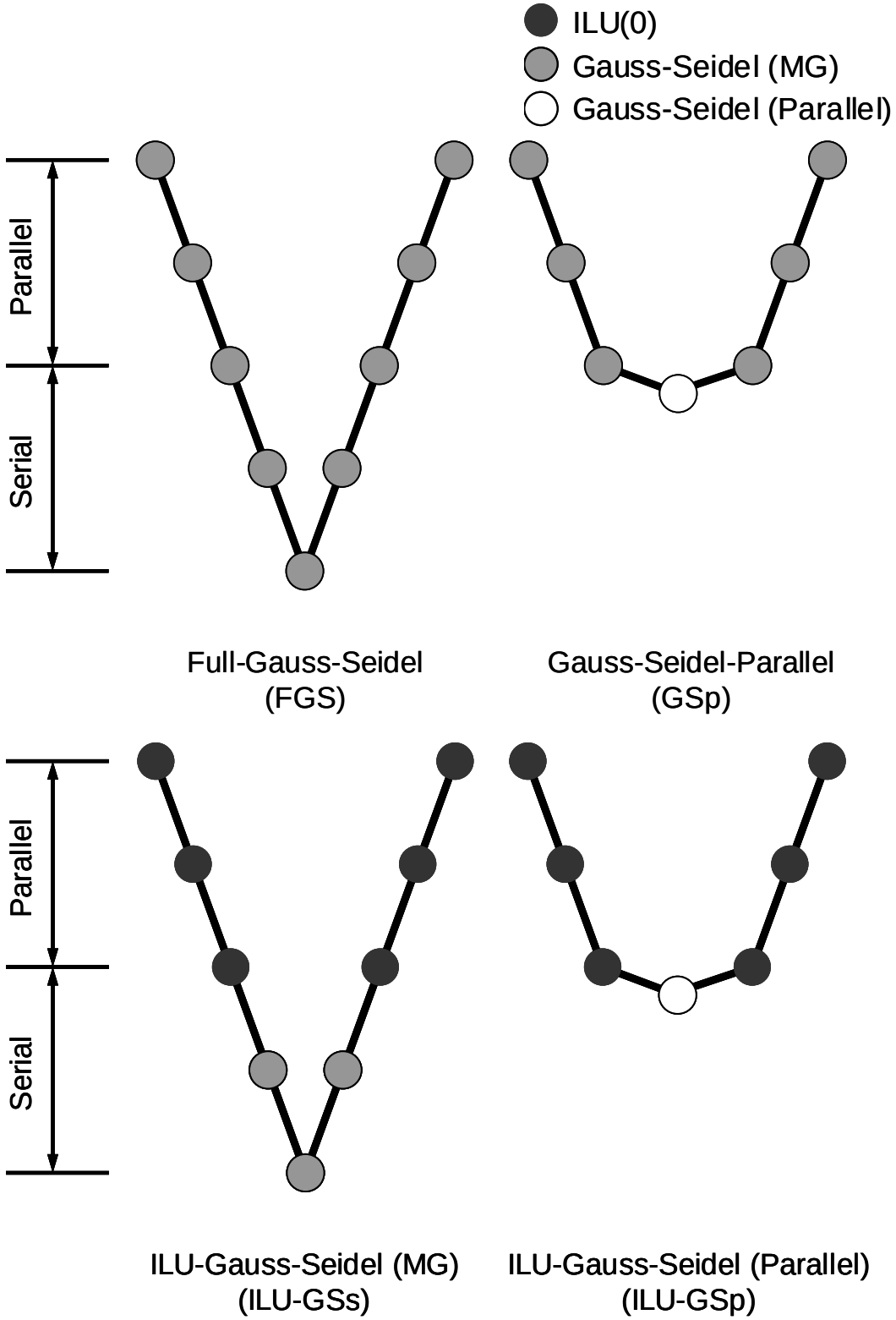


Fig. 5.15 Various types of multigrid procedures for parallel/serial operations

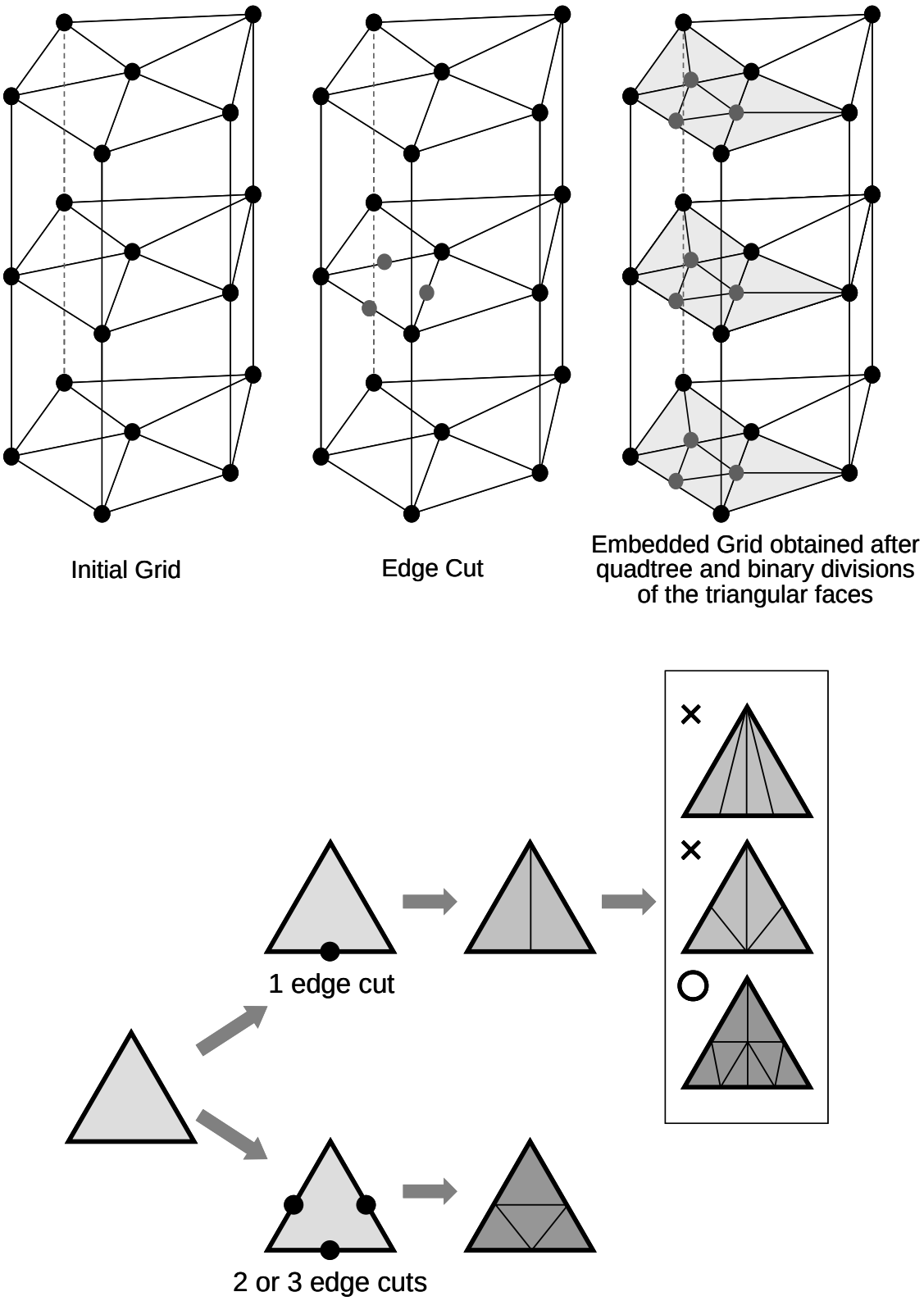


Fig. 5.16 Directional refinement of prisms based on *quadtree* and *binary* divisions of triangular faces [74,76,78,80,89,91]

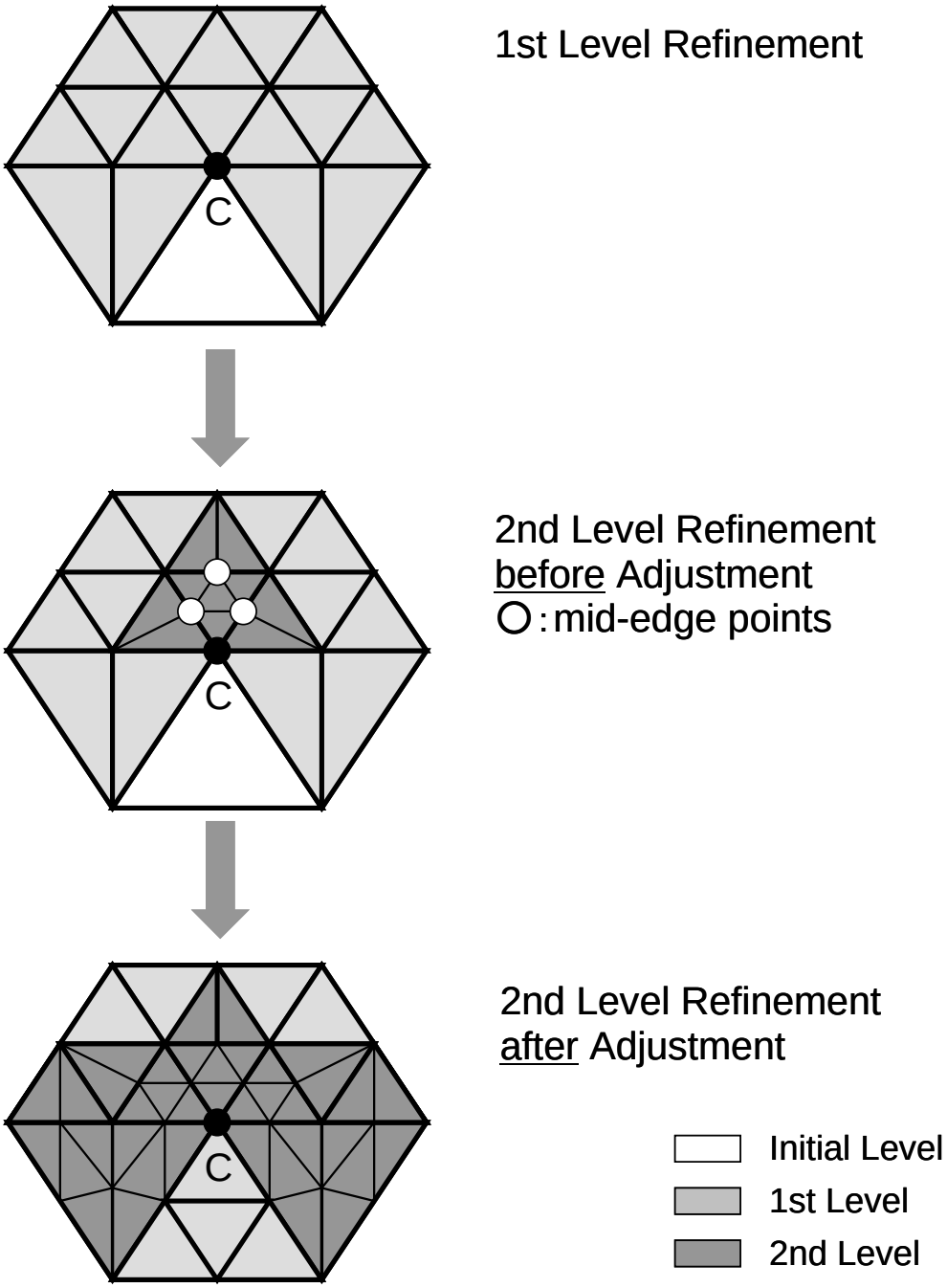


Fig. 5.17 Procedure in for avoiding sudden changes in mesh size. The mesh refinement algorithm limits the maximum difference in embedding level between neighboring elements to less than 2. [74,76,78,80,89,91]

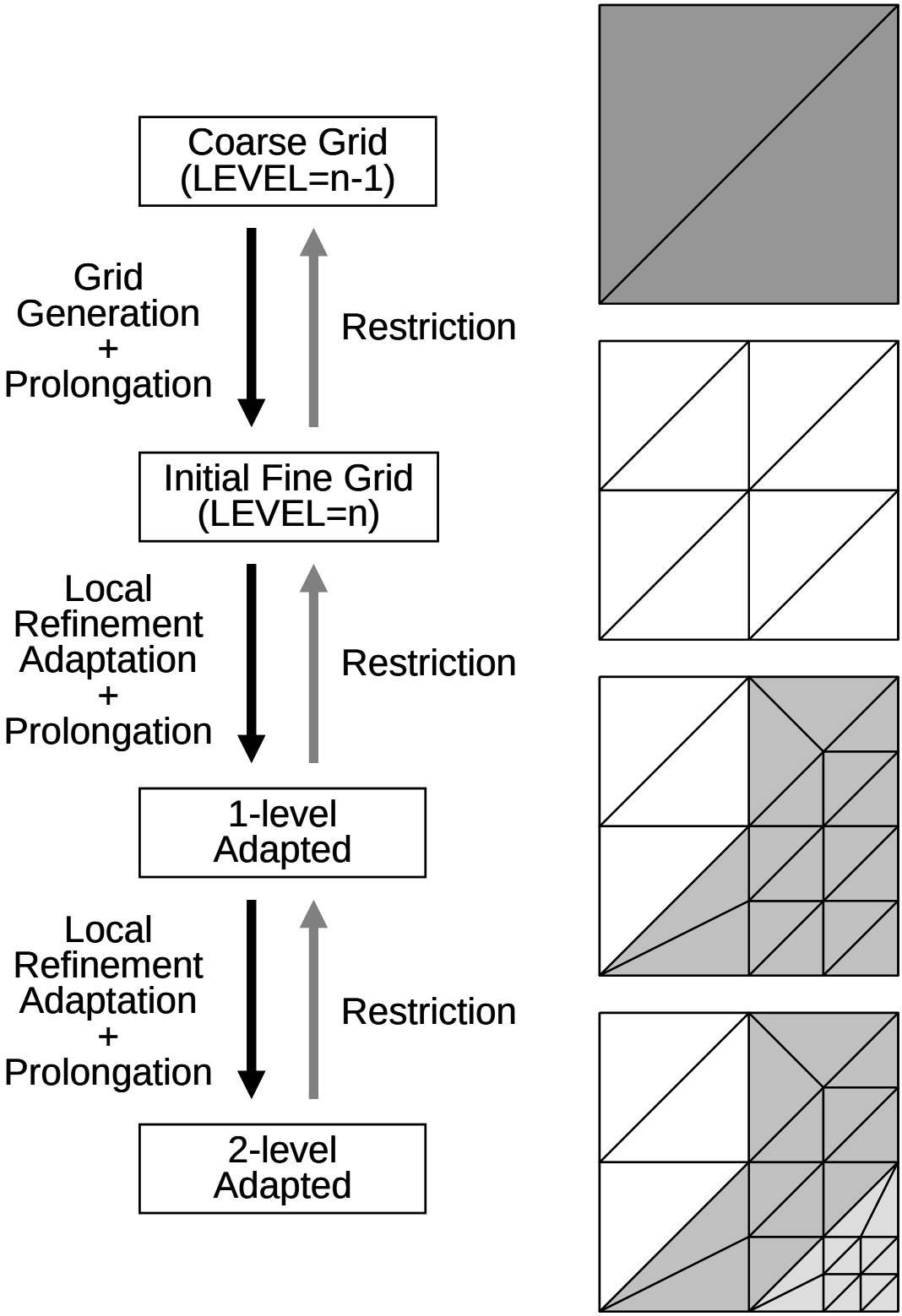


Fig. 5.18 Multigrid strategy for locally refined grids
Level-by-level and direct jump method [74,78,80]

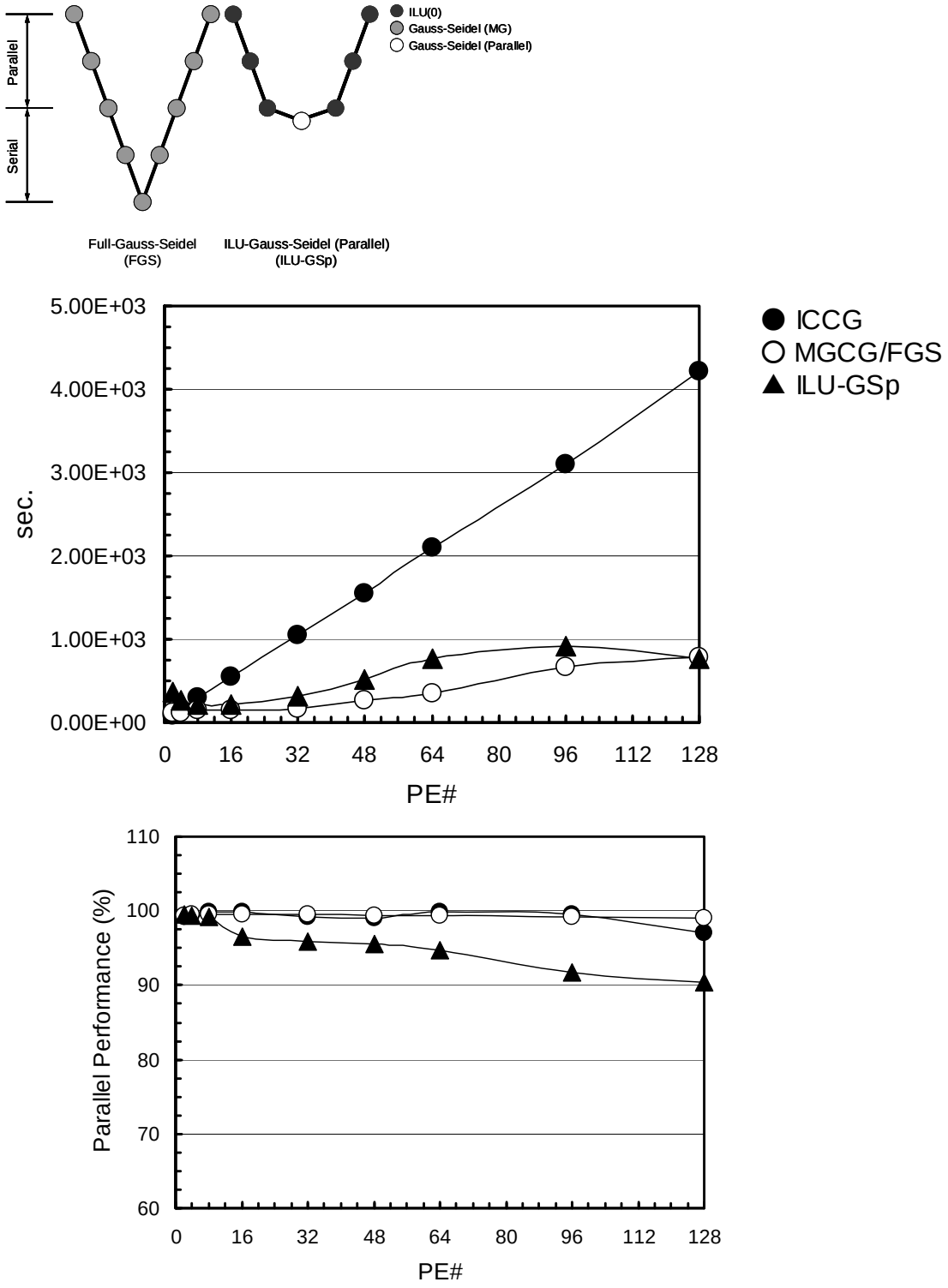


Fig. 5.19 Results of Poisson-I. Computation time (including communication for parallel computing) and parallel performance for fixed problem size on each processor (320×900=288,000 cells/PE) for 2 to 128 PEs (up to 36,864,000 cells). Uniform B.C. (Black Circles: ICCG, White Circles: MGCG/FGS, Black Tri.: MGCG/ILU-GSp)

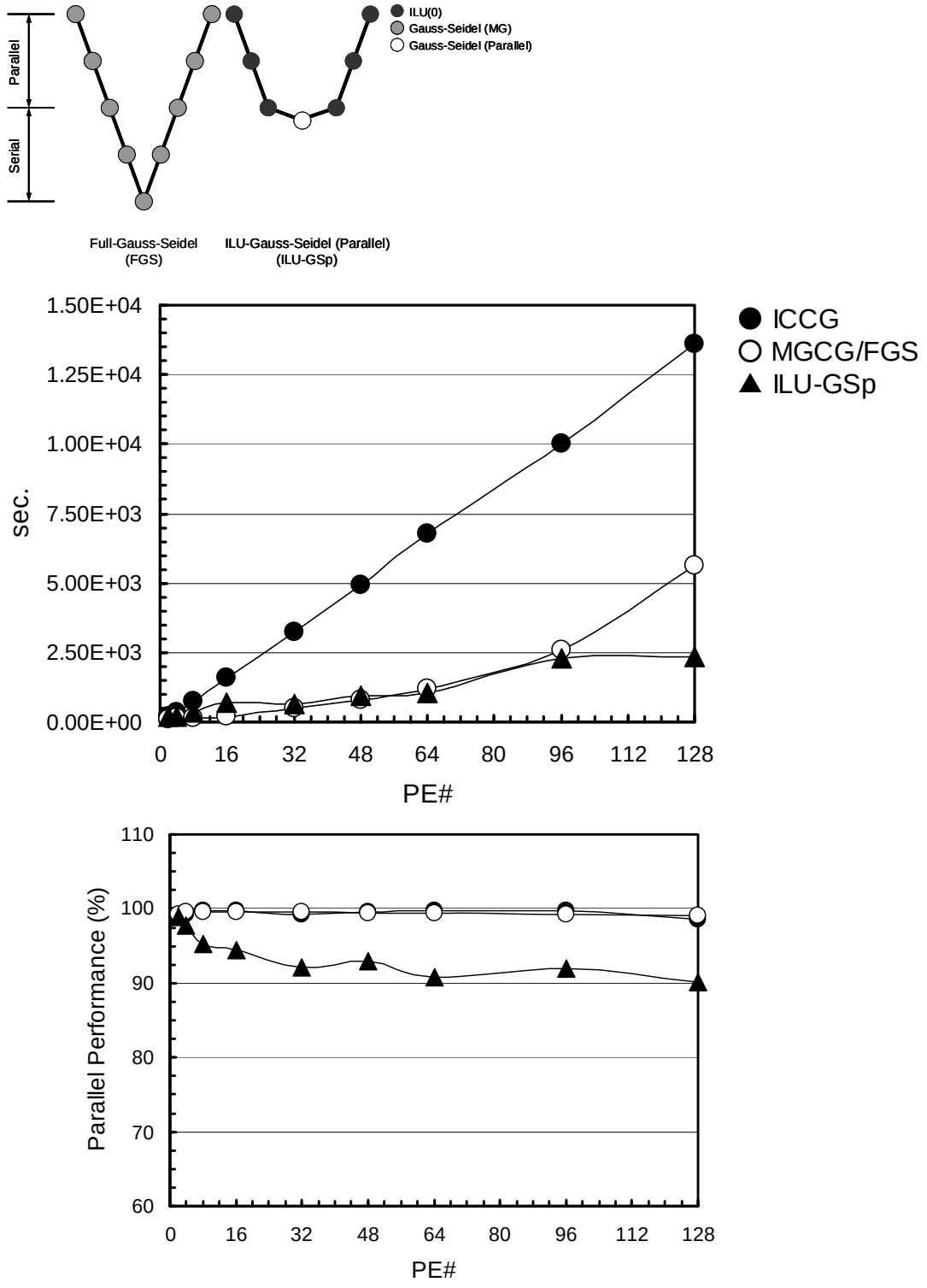


Fig. 5.20 Results of Poisson-I. Computation time (including communication for parallel computing) and parallel performance for fixed problem size on each processor (320×900=288,000 cells/PE) for 2 to 128 PEs (up to 36,864,000 cells). Single-Patch B.C. (Black Circles: ICCG, White Circles: MGCG/FGS, Black Tri.: MGCG/ILU-GSp)

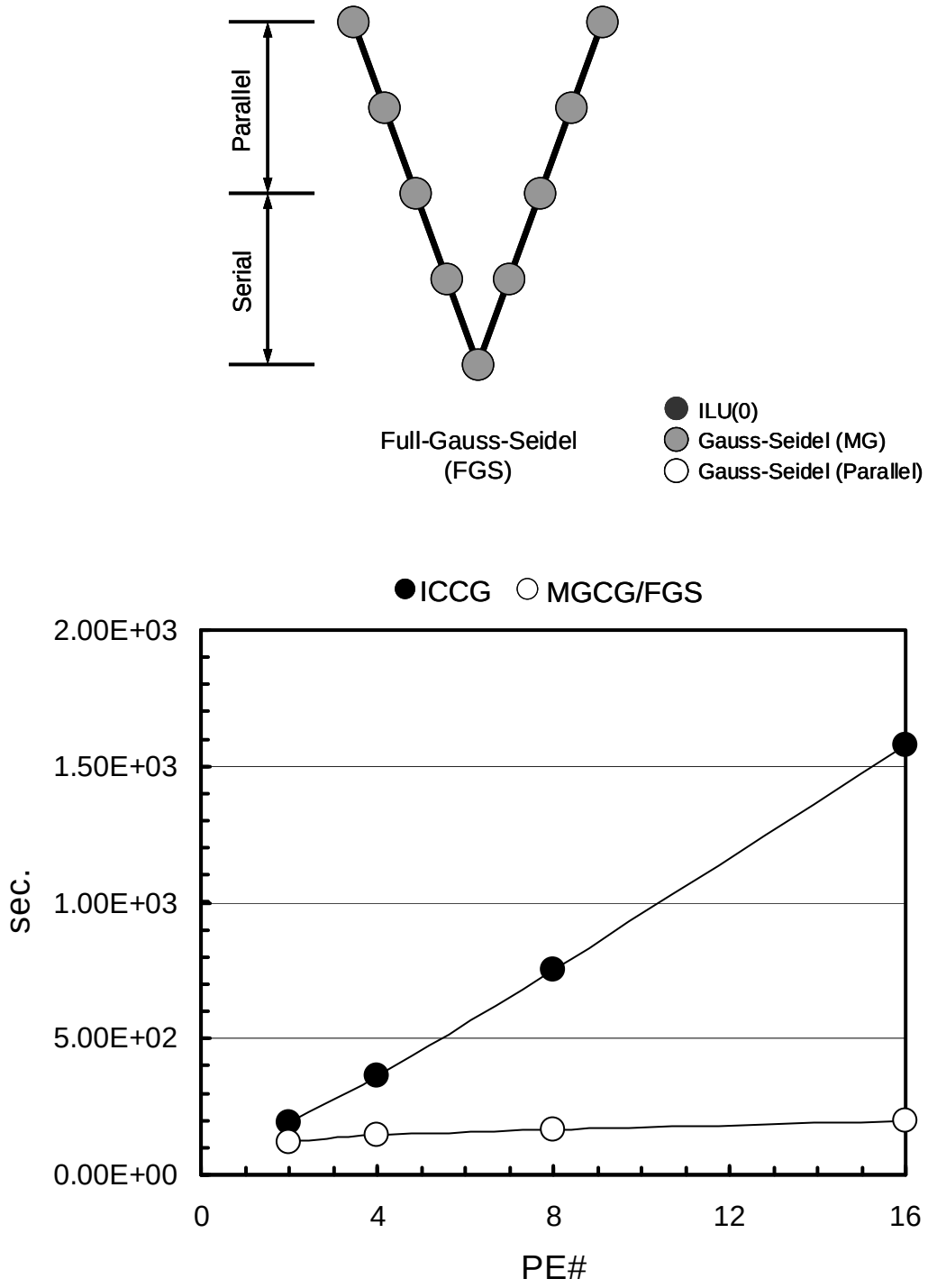
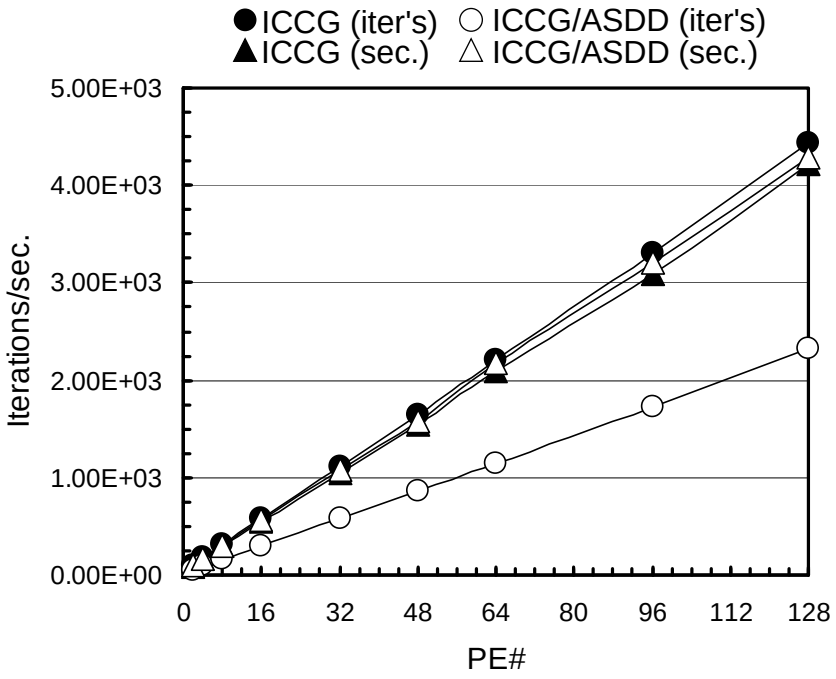


Fig. 5.21 Results of Poisson-I. Computation time (including communication for parallel computing) for fixed problem size on each processor (320×900=288,000 cells/PE) for 2 to 16 PEs (up to 4,608,000 cells). Single-Patch B.C. (Black Circles: ICCG, White Circles: MGCG/FGS)

(a) Uniform B.C.



(b) Single-Patch B.C.

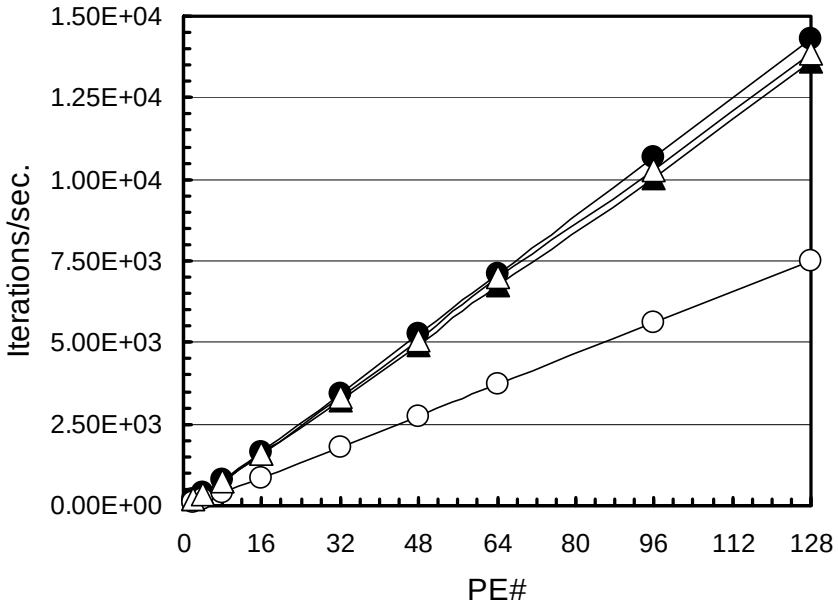


Fig. 5.22 Results of Poisson-I. Computation time (including communication for parallel computing) and iterations for convergence for fixed problem size on each processor (320×900=288,000 cells/PE) for 2 to 128 PEs (up to 36,864,000 cells). Comparison of ICCG and ICCG with Additive Schwartz Domain Decomposition (ASDD) (Black Circles/Tri.: ICCG iterations/comp.time, White Circles/Tri.: ICCG/ASDD iterations/comp.time)

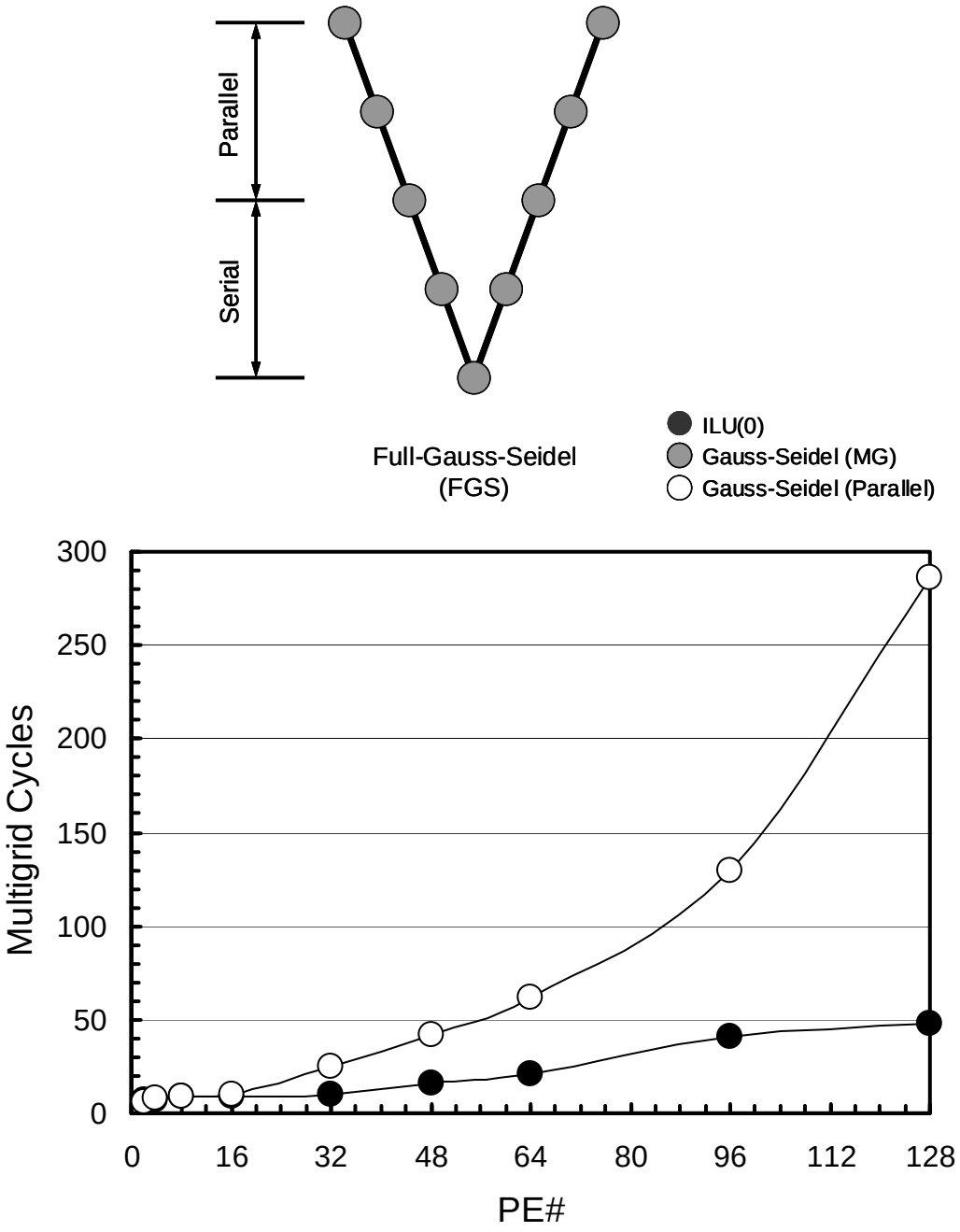


Fig. 5.23 Results of Poisson-I. Number of multigrid cycles for convergence for fixed problem size on each processor (320×900=288,000 cells/PE) for 2 to 128 PEs (up to 36,864,000 cells). Full Gauss Seidel Smoothing (FGS), (Black Circles: Uniform B.C., White Circles: One Patcch B.C.)

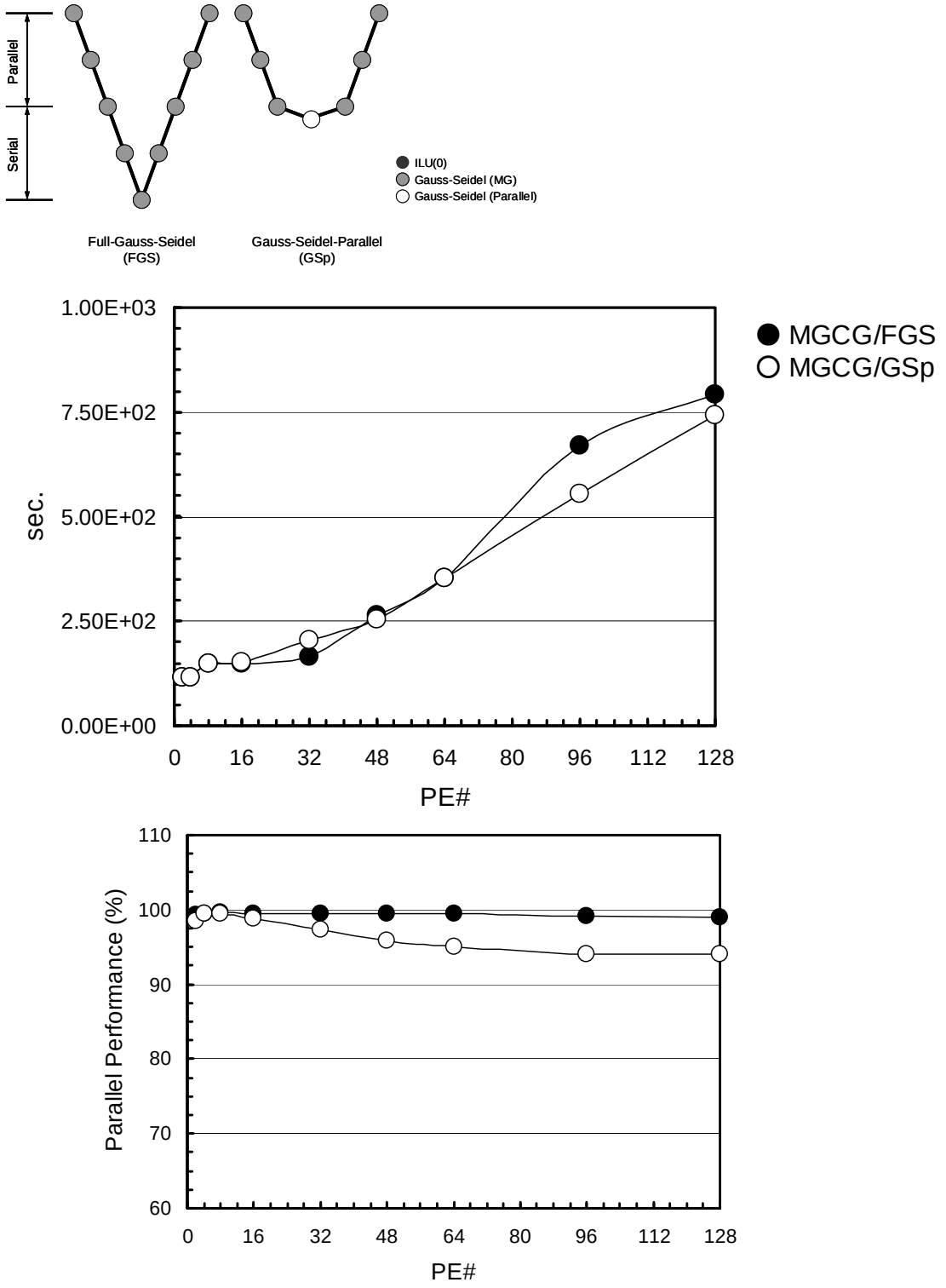


Fig. 5.24 Results of Poisson-I. Computation time (including communication for parallel computing) and parallel performance for fixed problem size on each processor (320×900=288,000 cells/PE) for 2 to 128 PEs (up to 36,864,000 cells). Uniform B.C. (Black Circles: MGCG/FGS, White Circles: MGCG/GSp)

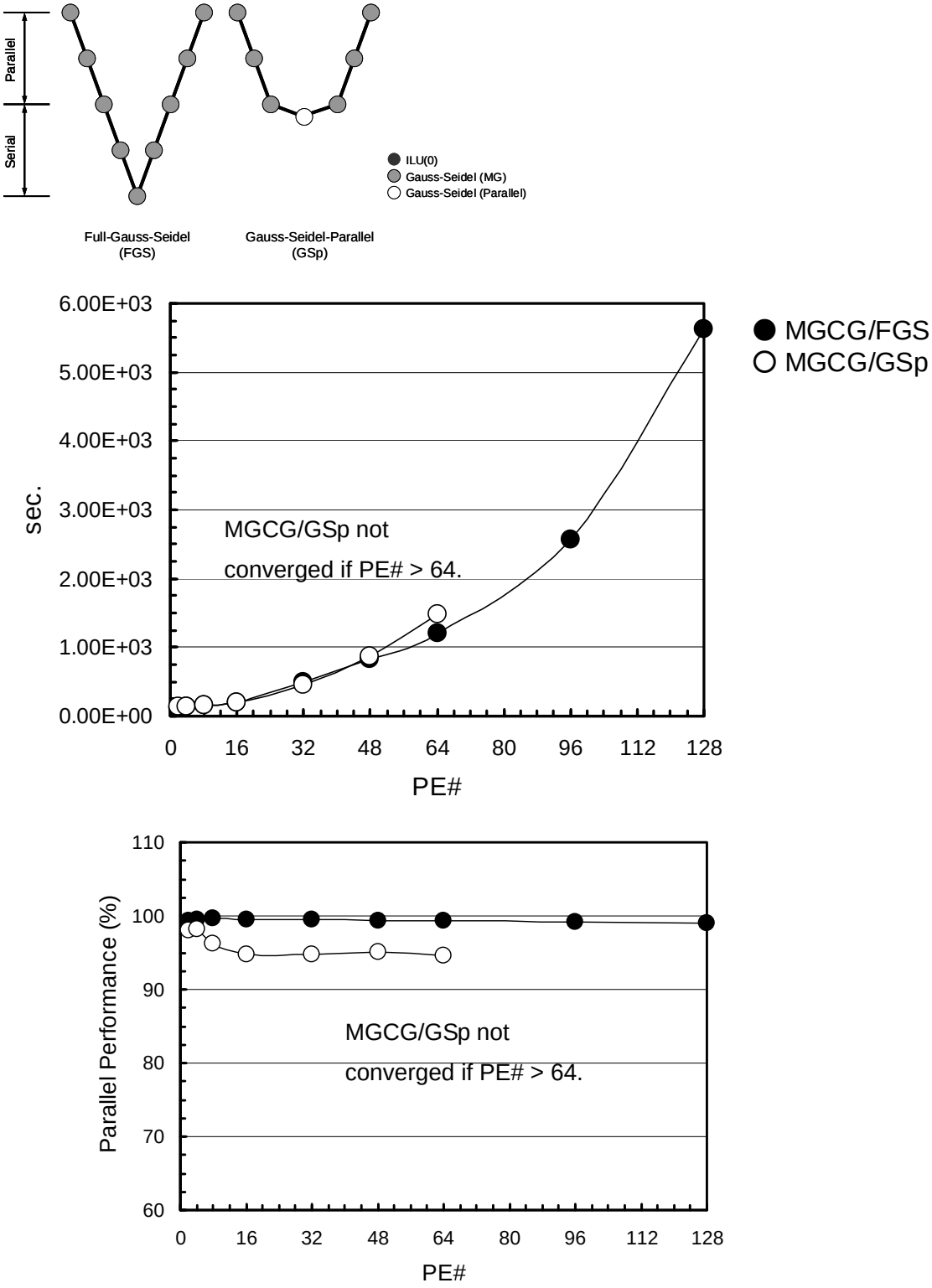


Fig. 5.25 Results of Poisson-I. Computation time (including communication for parallel computing) and parallel performance for fixed problem size on each processor (320×900=288,000 cells/PE) for 2 to 128 PEs (up to 36,864,000 cells). Single-Patch B.C. (Black Cir.: MGCG/FGS, White Cir.: MGCG/GSp)

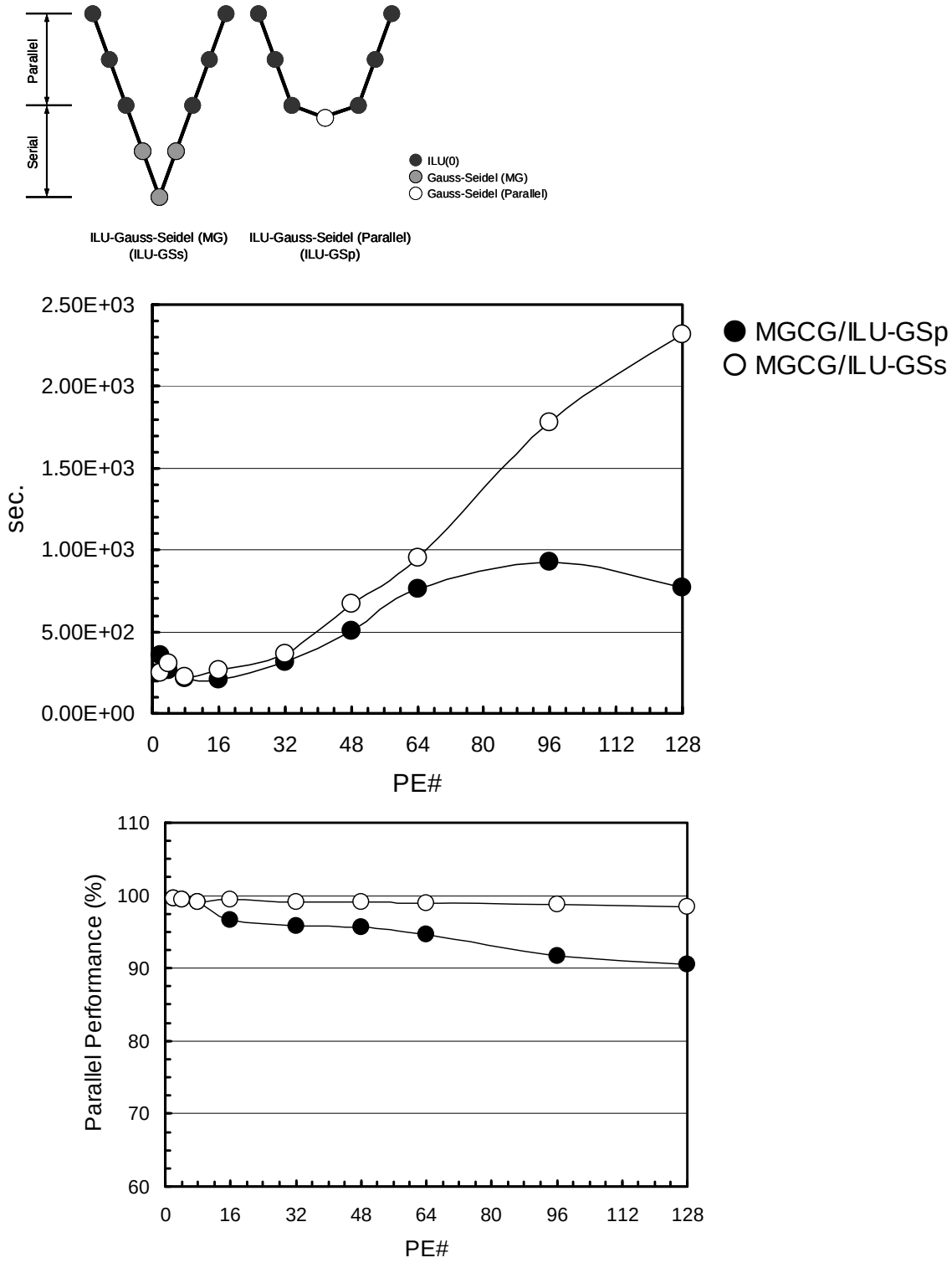


Fig. 5.26 Results of Poisson-I. Computation time (including communication for parallel computing) and parallel performance for fixed problem size on each processor (320×900=288,000 cells/PE) for 2 to 128 PEs (up to 36,864,000 cells). Uniform B.C. (Black Circles: MGCG/ILU-GSp, White Circles: MGCG/ILU-GSs)

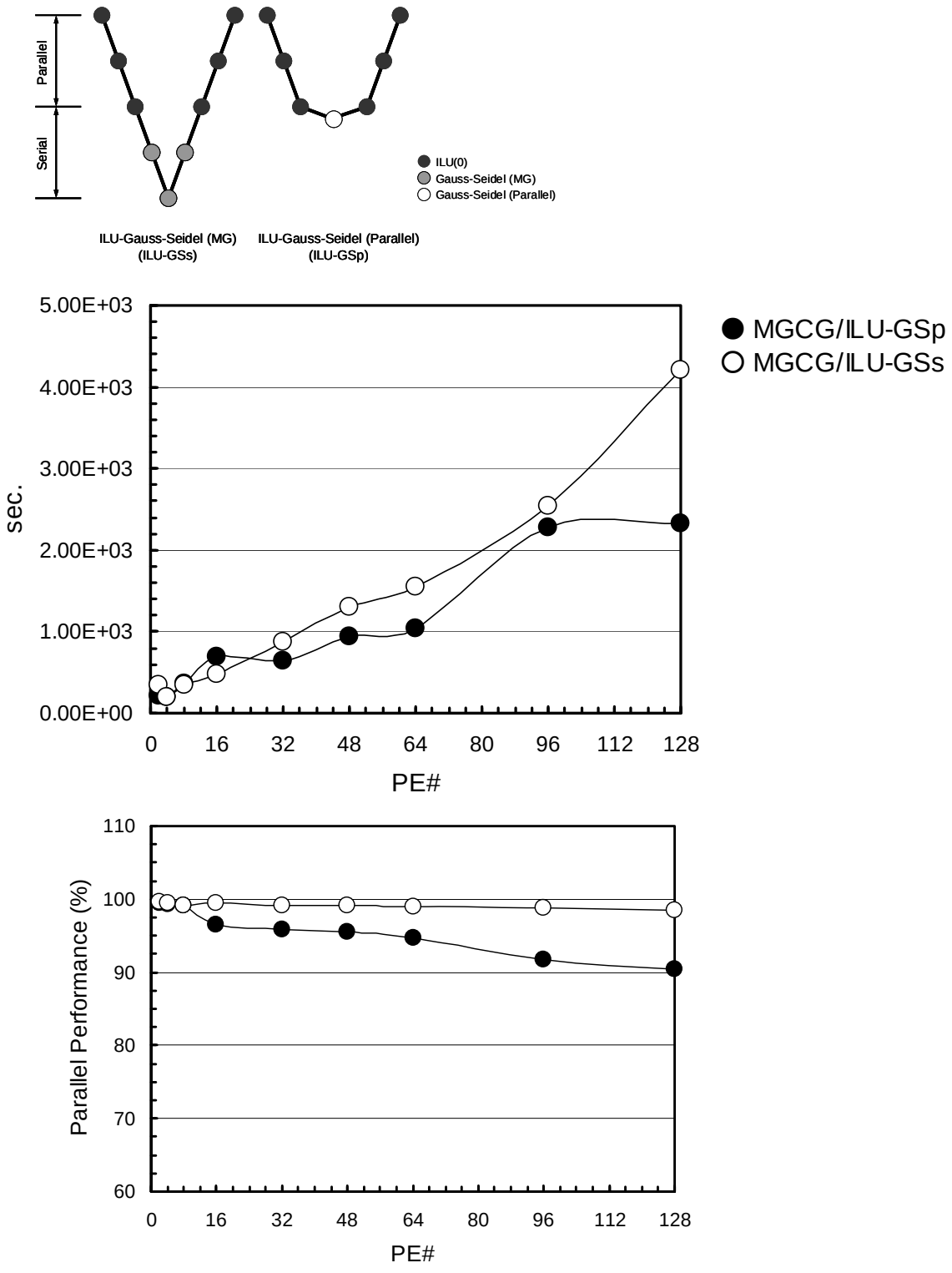


Fig. 5.27 Results of Poisson-I. Computation time (including communication for parallel computing) and parallel performance for fixed problem size on each processor (320×900=288,000 cells/PE) for 2 to 128 PEs (up to 36,864,000 cells). Single-Patch B.C. (Black Cir.: MGCG/ILU-GSp, White Cir.: MGCG/ILU-GSs)

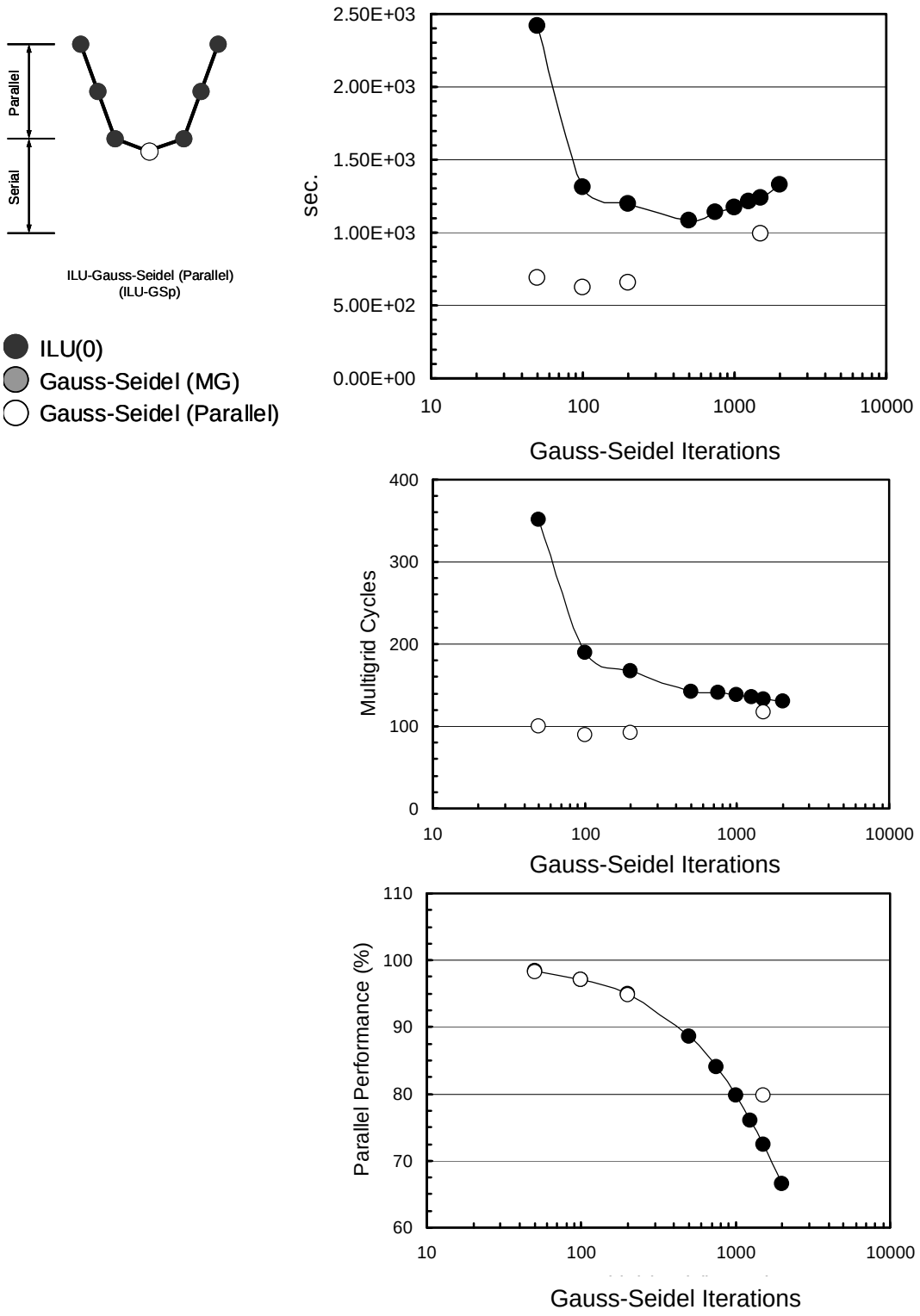


Fig. 5.28 Results of Poisson-I. Computation time (including communication for parallel computing), multigrid cycles and parallel performance for 64 PEs (18,432,000 cells). Effect of Gauss-Seidel iteration number of MGCG/ILU-GSp. (Black Circles: Uniforma B.C., White Circles: Single-Patch B.C.)

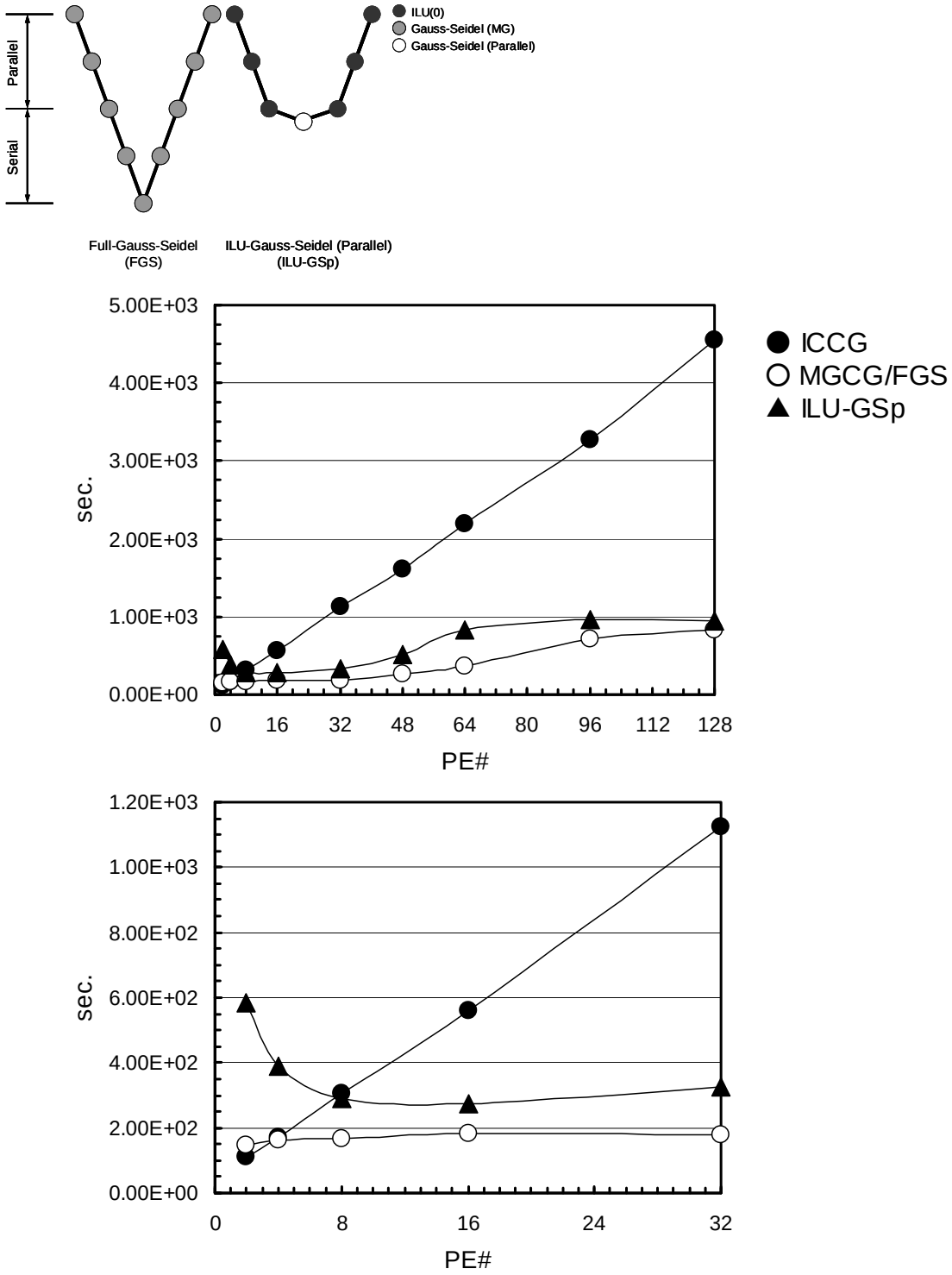


Fig. 5.29 Results of Poisson-I. for CLUSTERED mesh in radial direction. Computation time (including communication for parallel computing) for fixed problem size on each processor (320×900=288,000 cells/PE) for 2 to 128 PEs (up to 36,864,000 cells). Uniform B.C. (Black Circles: ICCG, White Circles: MGCG/FGS, Black Tri.: MGCG/ILU-GSp)

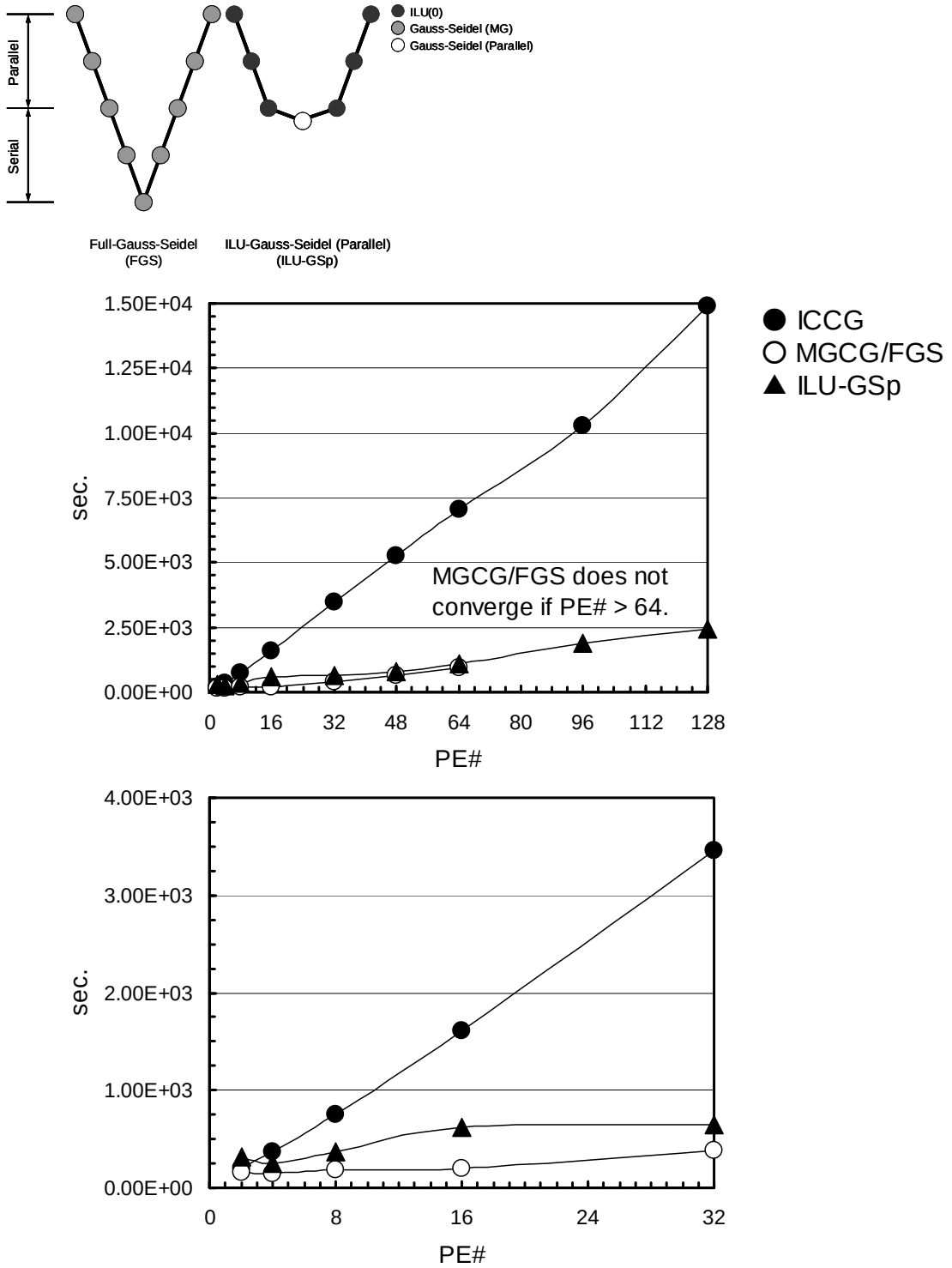
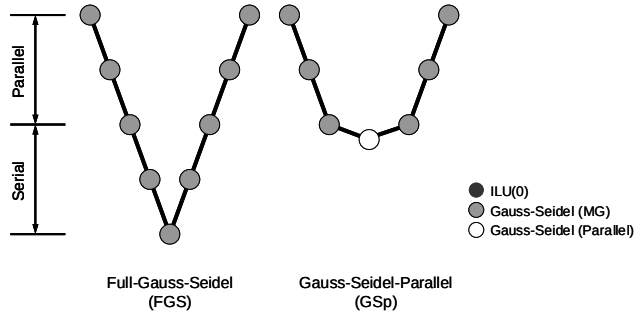
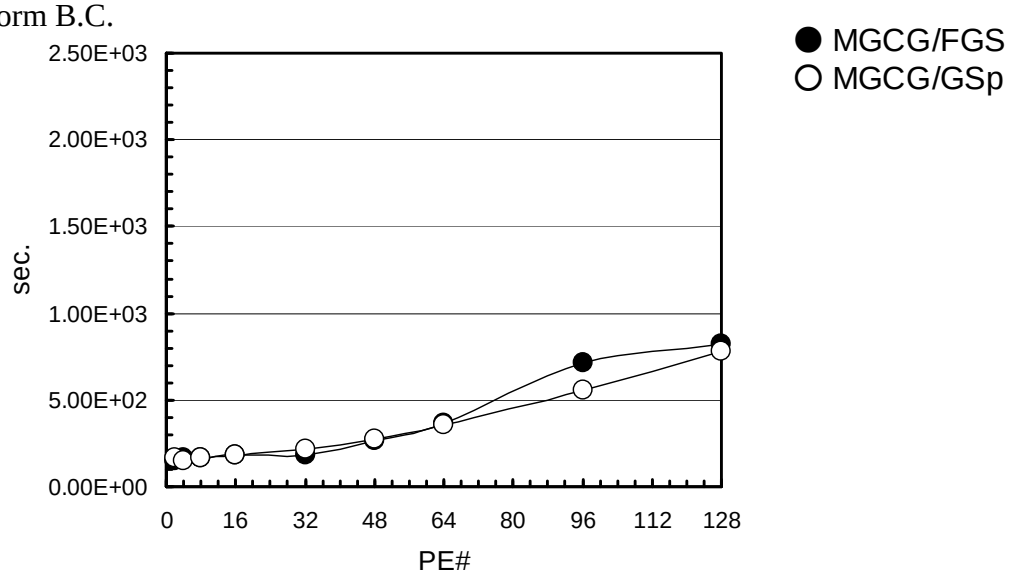


Fig. 5.30 Results of Poisson-I. for CLUSTERED mesh in radial direction. Computation time (including communication for parallel computing) for fixed problem size on each processor (320×900=288,000 cells/PE) for 2 to 128 PEs (up to 36,864,000 cells). Single-Patch B.C. (Black Circles: ICCG, White Circles: MGCG/FGS, Black Tri.: MGCG/ILU-GSp)



(a) Uniform B.C.



(b) One-Patch B.C.

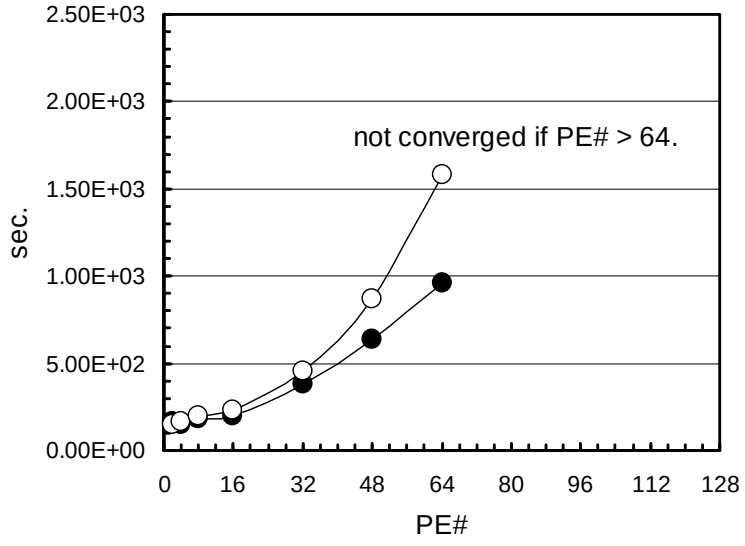
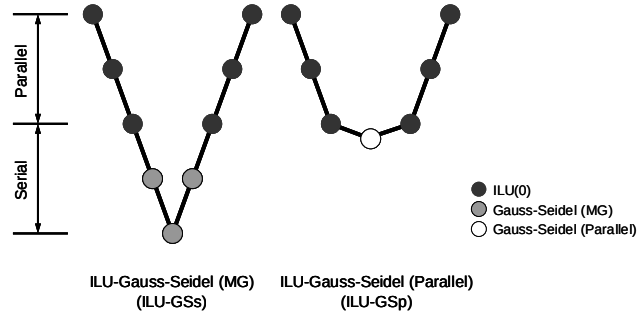
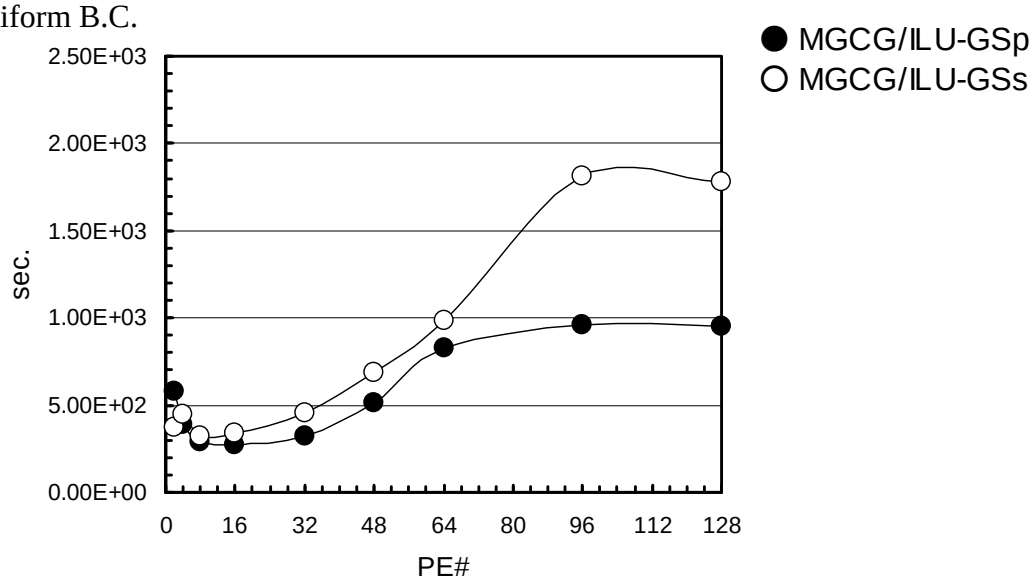


Fig. 5.31 Results of Poisson-I. for CLUSTERED mesh in radial direction. Computation time (including communication for parallel computing) for fixed problem size on each processor (320×900=288,000 cells/PE) for 2 to 128 PEs (up to 36,864,000 cells) (Black Circles: MGCG/FGS, White Circles: MGCG/GSp).



(a) Uniform B.C.



(b) One-Patch B.C.

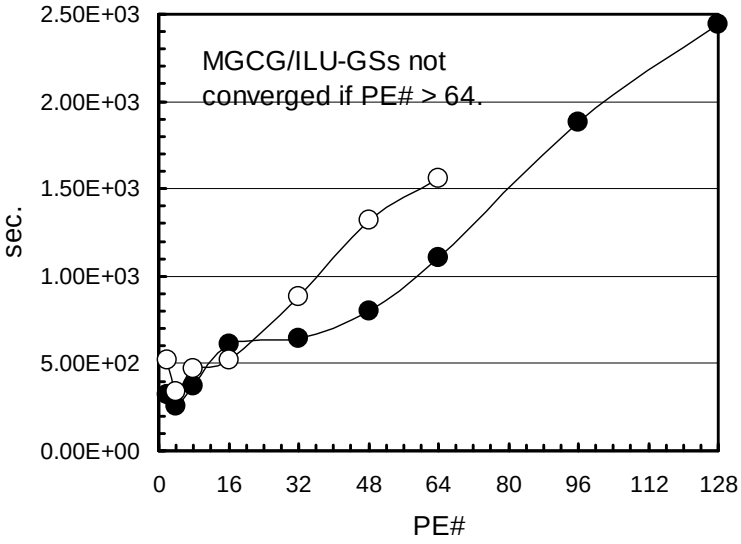


Fig. 5.32 Results of Poisson-I. for CLUSTERED mesh in radial direction. Computation time (including communication for parallel computing) for fixed problem size on each processor (320×900=288,000 cells/PE) for 2 to 128 PEs (up to 36,864,000 cells) (Black Circles: MGCG/ILU-GSp, White Circles: MGCG/ILU-GSs).

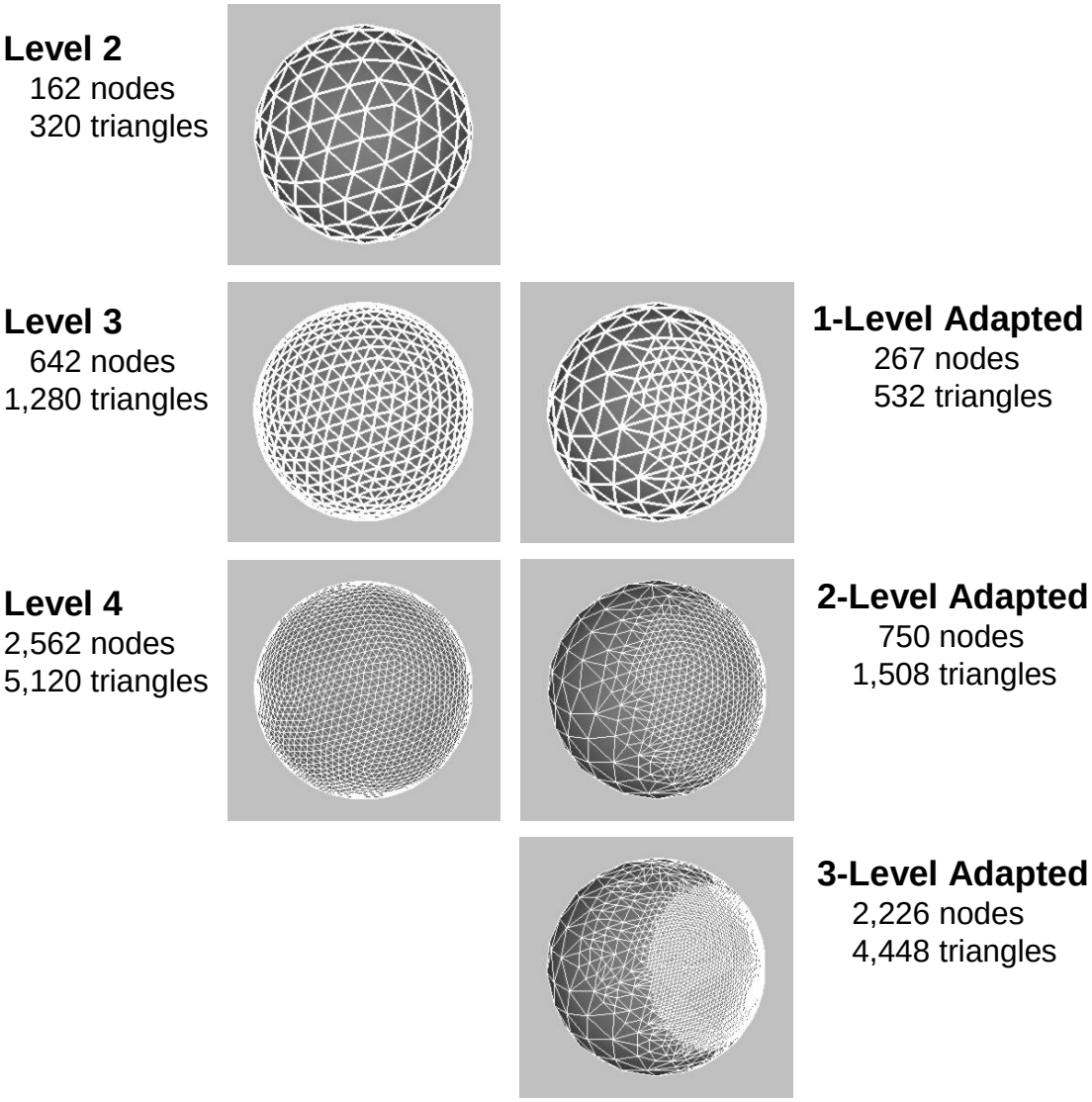


Fig. 5.33 Locally refined surface meshes

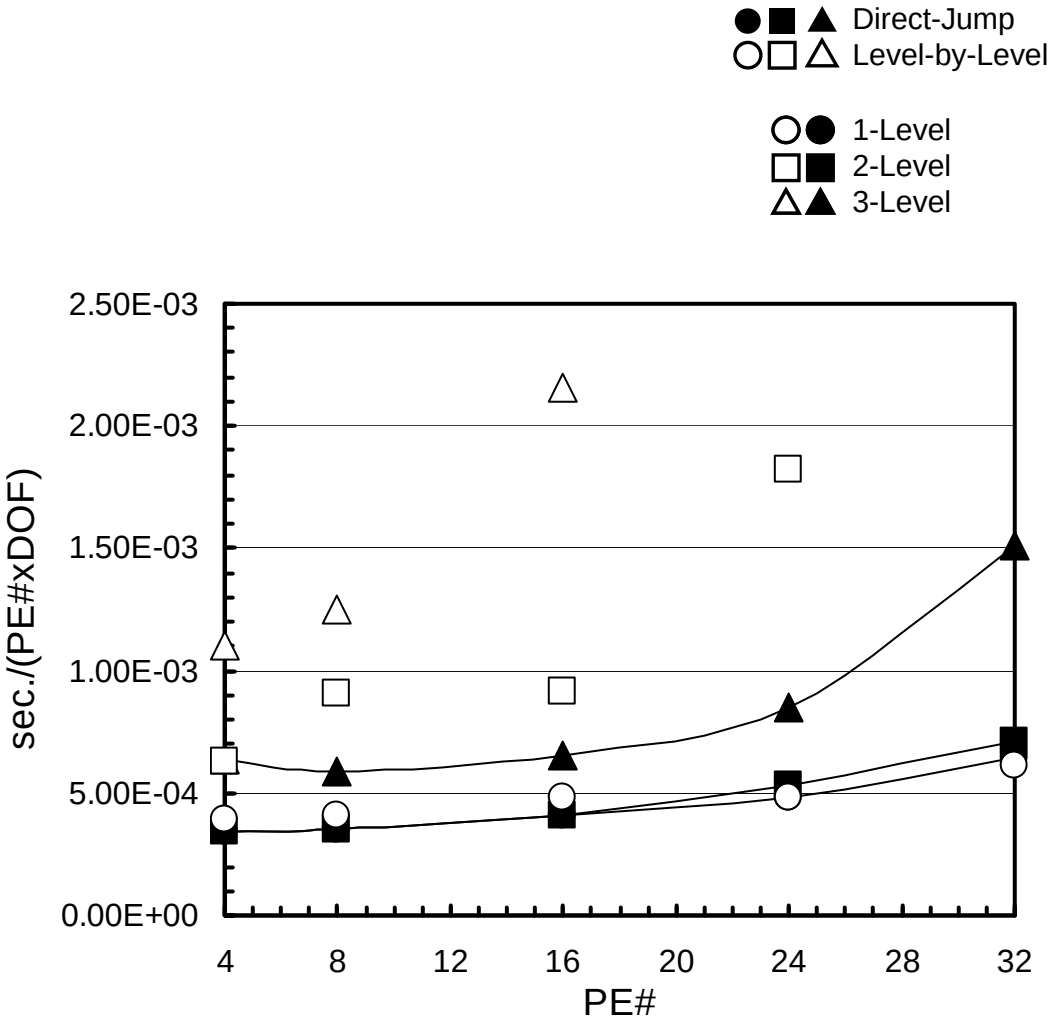
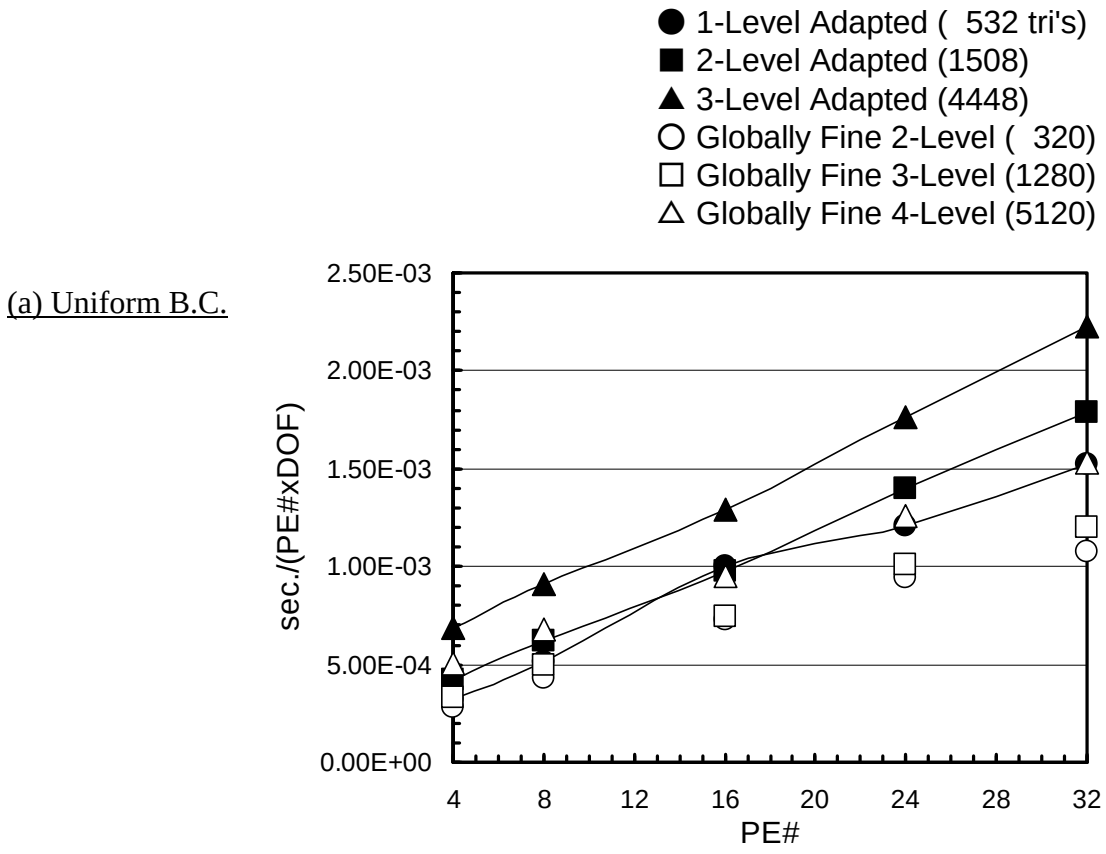


Fig. 5.34 Results of Poisson-II. Computation time (including communication, normalized by cell number/PE for parallel computing) for locally refined grids using MGCG/FGS, 50 layers/PE, for 4 to 32 PEs (up to 7,116,800 cells). Uniform B.C.
(Solid line: initial grid (320 triangles), circles: 1-level adapted, squares: 2-level adapted, triangles: 3-level adapted.)



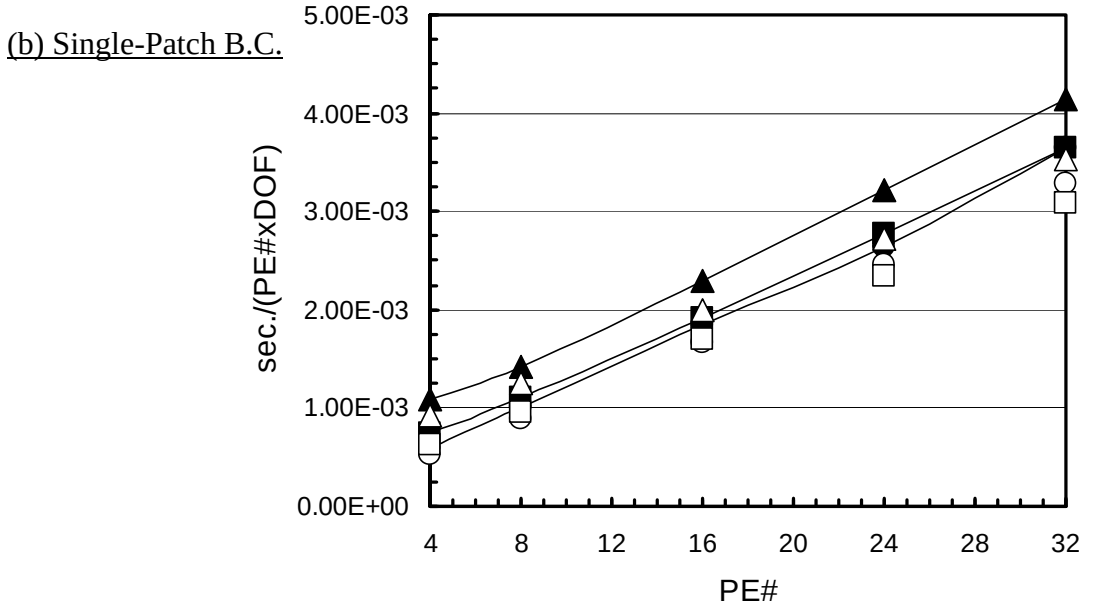


Fig. 5.35 Results of Poisson-II. Computation time (including communication, normalized by cell number/PE for parallel computing) for locally refined grids using ICCG, 50 layers/PE, for 4 to 32 PEs (up to 7,116,800 cells). Uniform & Single-Patch B.C., Direct Jump.

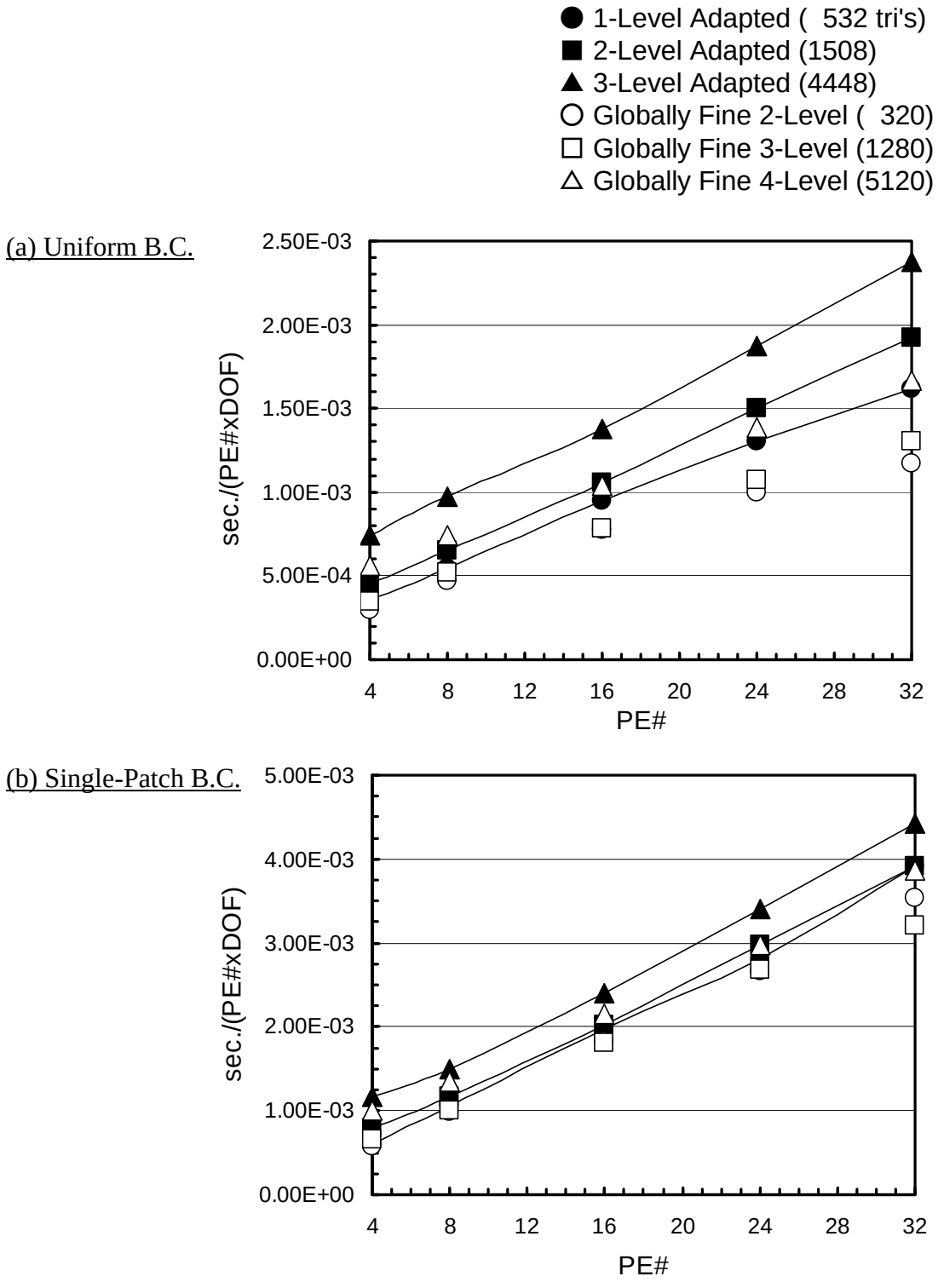


Fig. 5.36 Results of Poisson-II. Computation time (including communication, normalized by cell number/PE for parallel computing) for locally refined grids using ICCG/ASDD, 50 layers/PE, for 4 to 32 PEs (up to 7,116,800 cells). Uniform & Single-Patch B.C., Direct Jump.

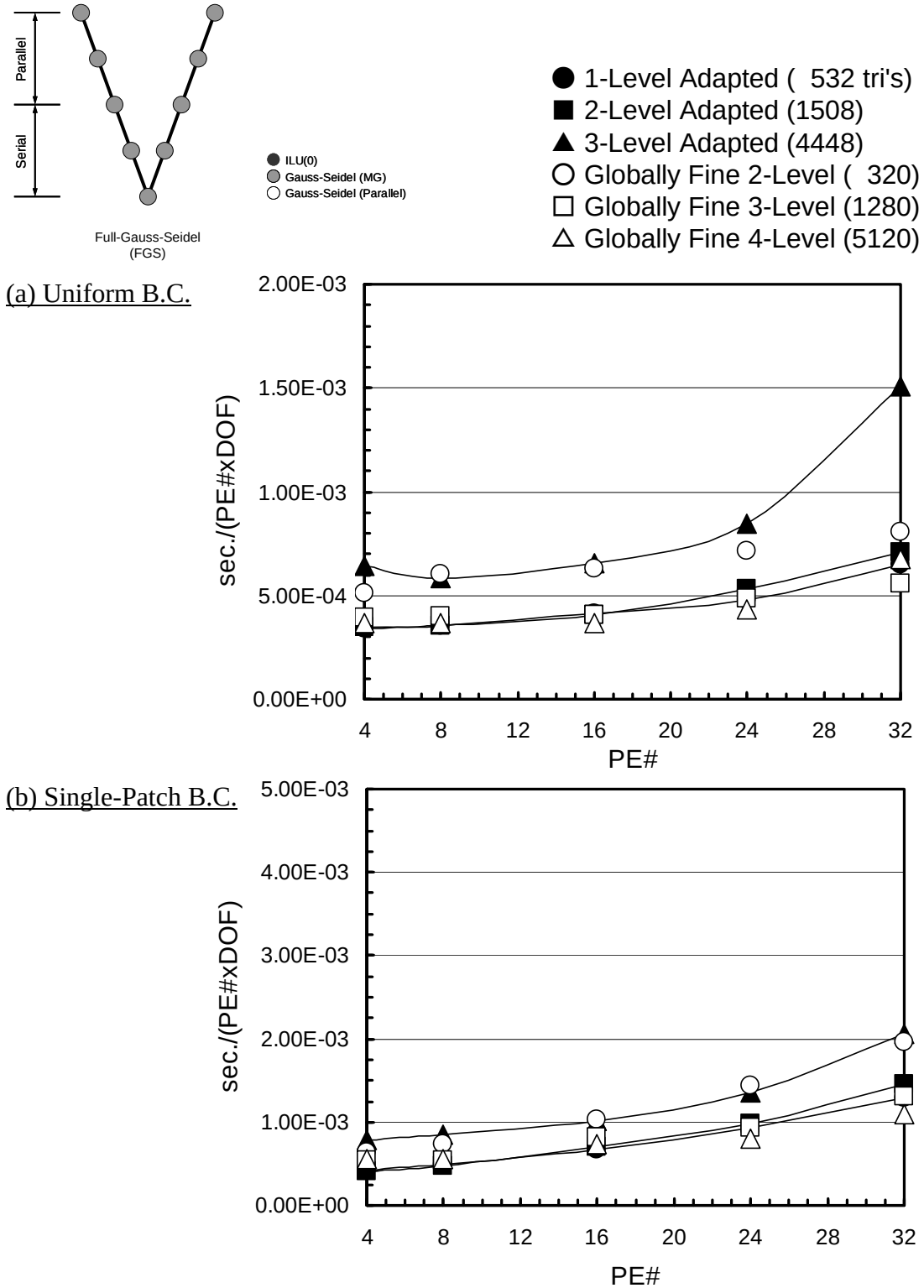


Fig. 5.37 Results of Poisson-II. Computation time (including communication, normalized by cell number/PE for parallel computing) for locally refined grids using MGCG/FGS, 50 layers/PE, for 4 to 32 PEs (up to 7,116,800 cells). Uniform & Single-Patch B.C., Direct Jump.

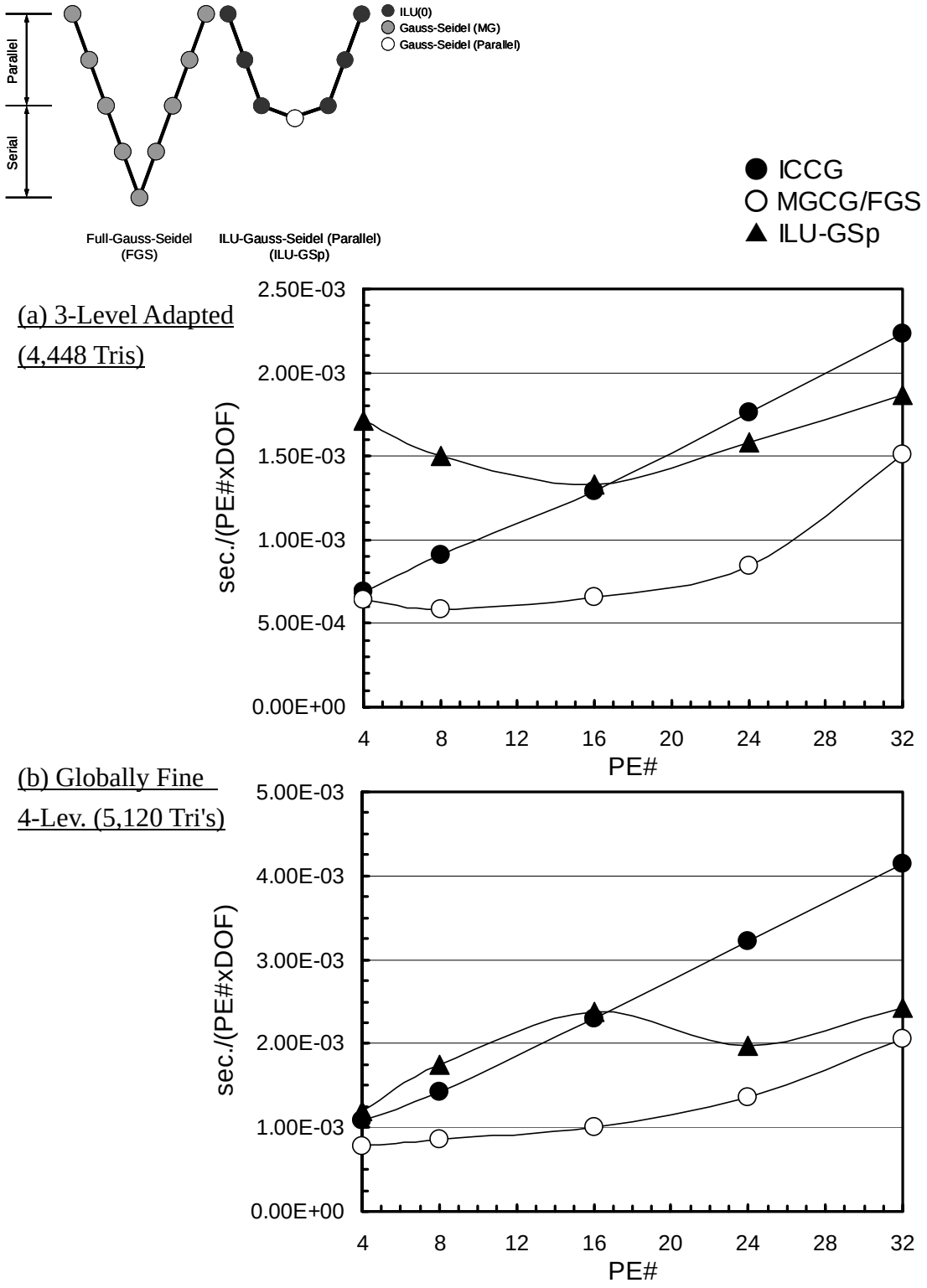


Fig. 5.38 Results of Poisson-II. Computation time (including communication, normalized by cell number/PE for parallel computing) for locally refined grids, 50 layers/PE, for 4 to 32 PEs (up to 7,116,800 cells). Uniform B.C., Direct Jump.
(Black Circles: ICCG, White Circles: MGCG/FGS, Black Triangles: MGCG/ILU-GSp)

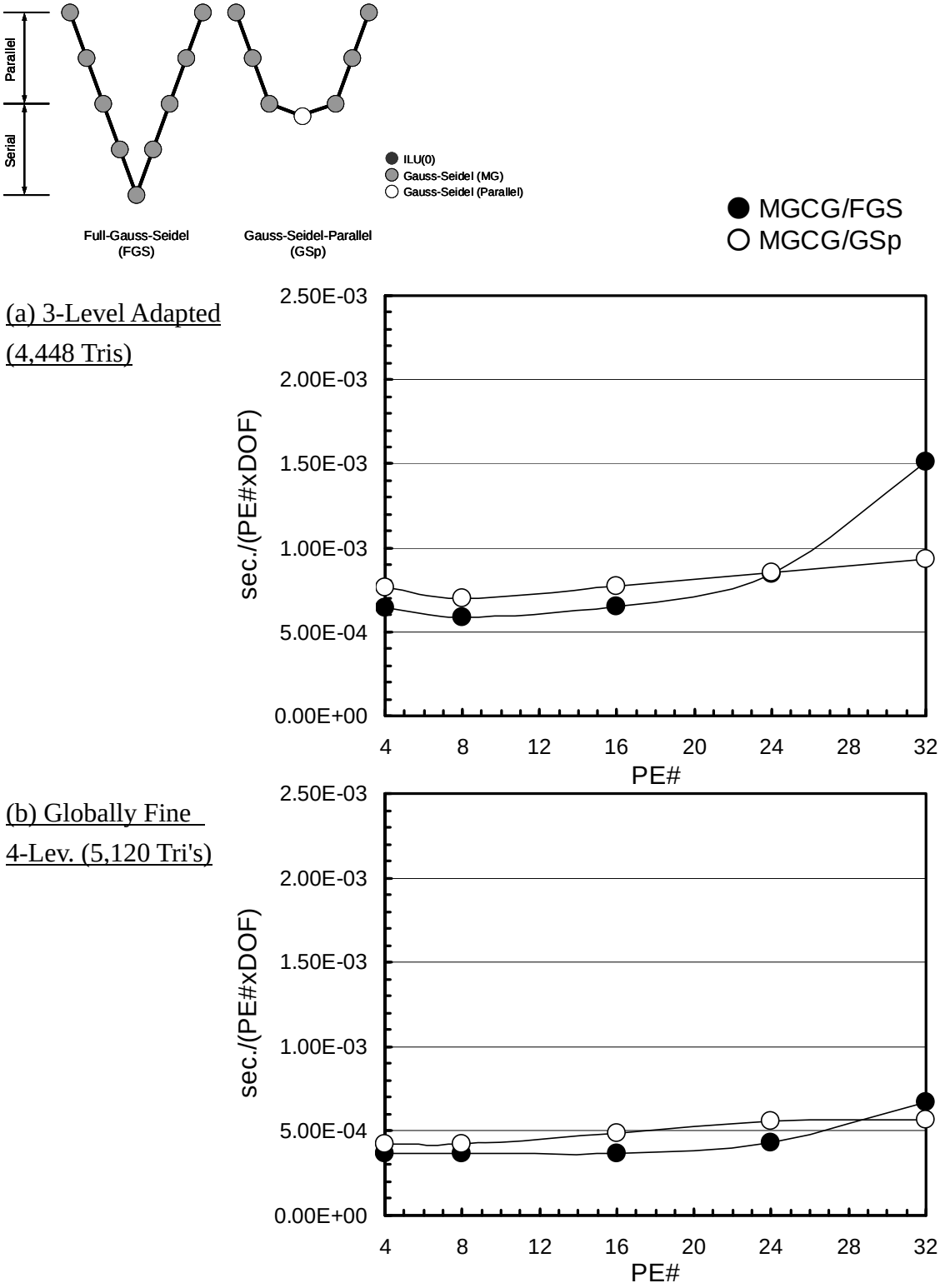


Fig. 5.39 Results of Poisson-II. Computation time (including communication, normalized by cell number/PE for parallel computing) for locally refined grids, 50 layers/PE, for 4 to 32 PEs (up to 7,116,800 cells). Uniform B.C., Direct Jump.
(Black: MGCG/FGS, White: MGCG/GSp)

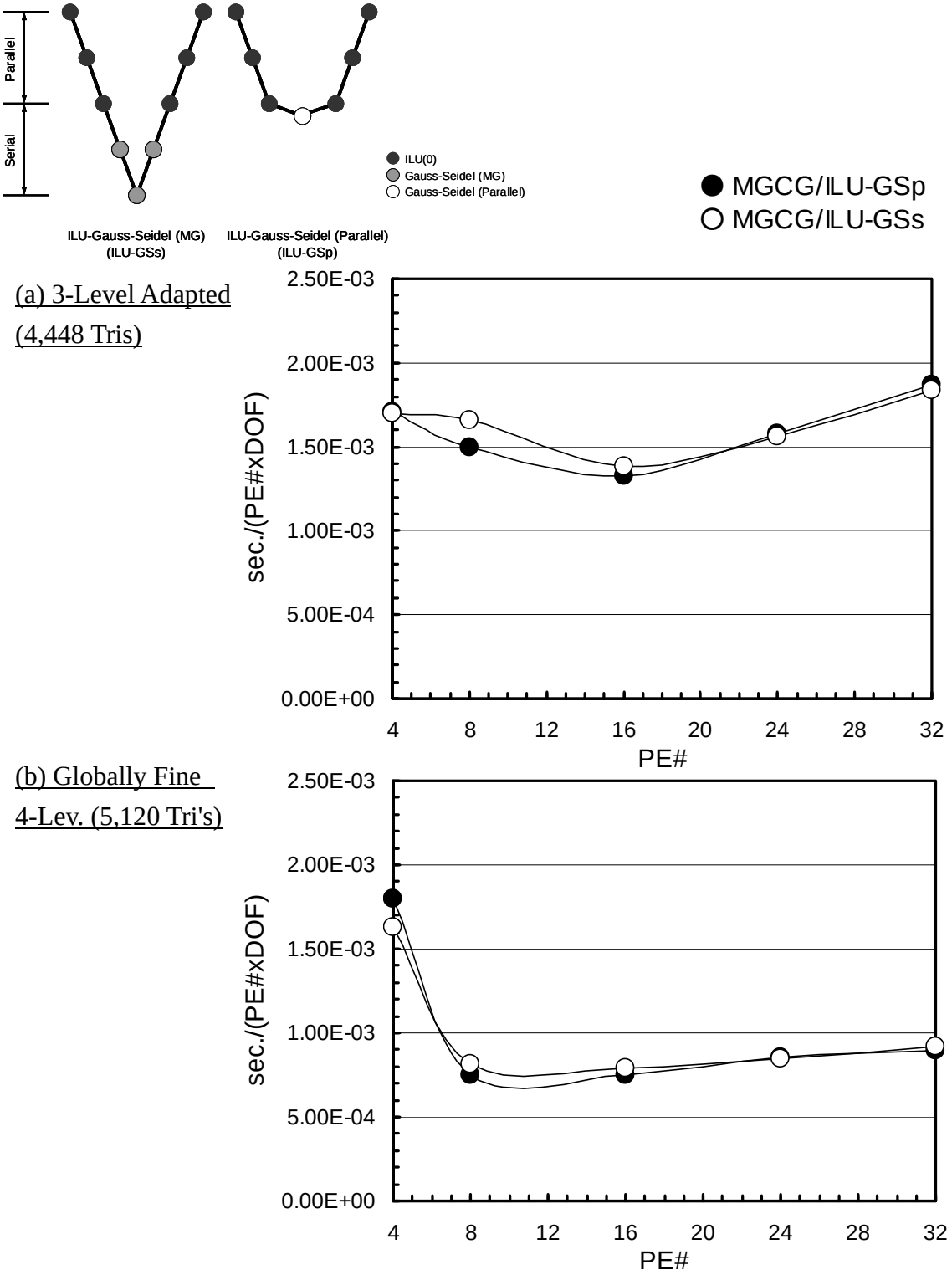


Fig. 5.40 Results of Poisson-II. Computation time (including communication, normalized by cell number/PE for parallel computing) for locally refined grids, 50 layers/PE, for 4 to 32 PEs (up to 7,116,800 cells). Uniform B.C., Direct Jump.
(Black: MGCG/ILU-GSp, White.: MGCG/ILU-GSs)

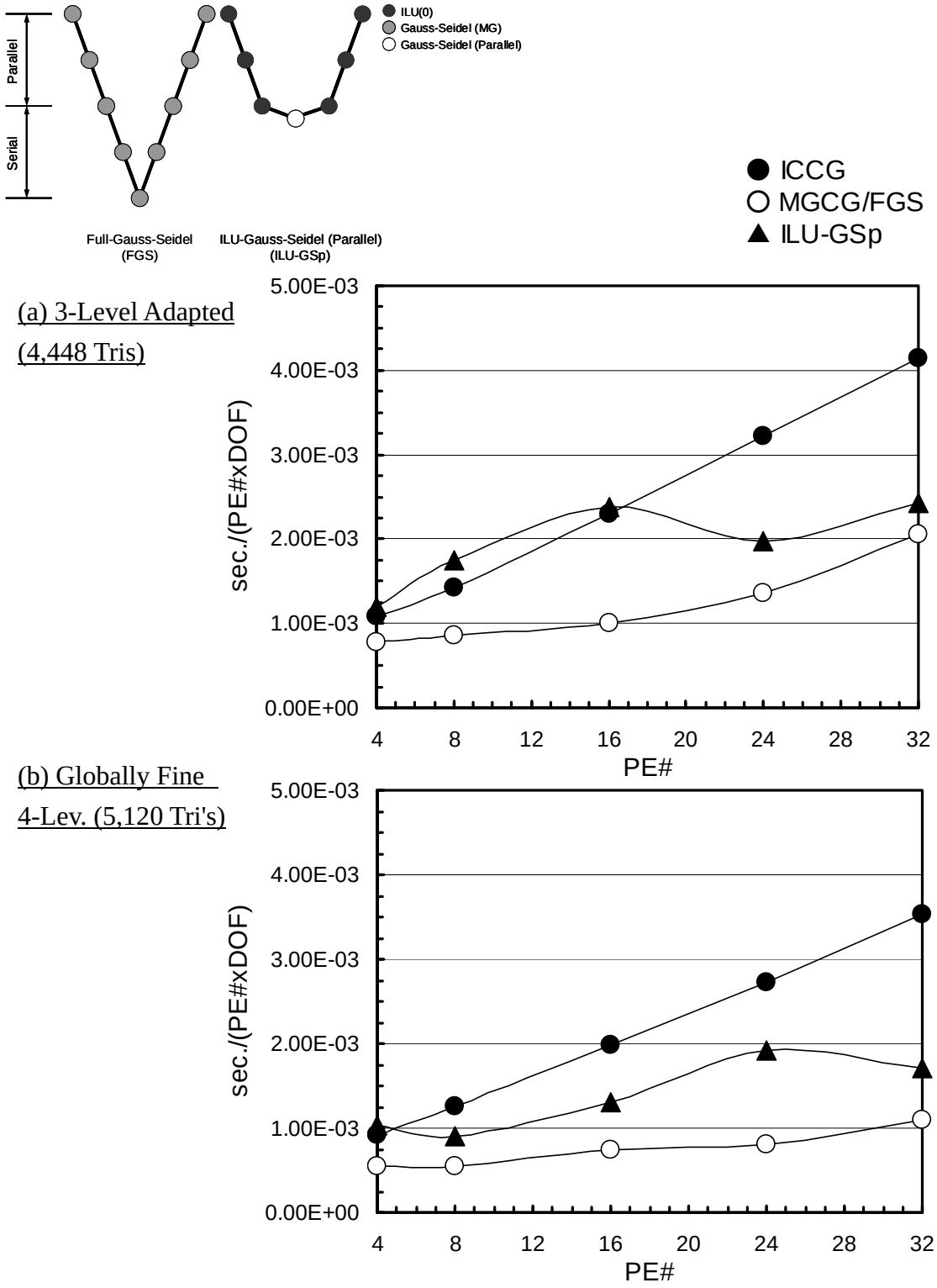


Fig. 5.41 Results of Poisson-II. Computation time (including communication, normalized by cell number/PE for parallel computing) for locally refined grids, 50 layers/PE, for 4 to 32 PEs (up to 7,116,800 cells). Single-Patch B.C., Direct Jump. (Black Circles: ICCG, White Circles: MGCG/FGS, Black Triangles: MGCG/ILU-GSp)

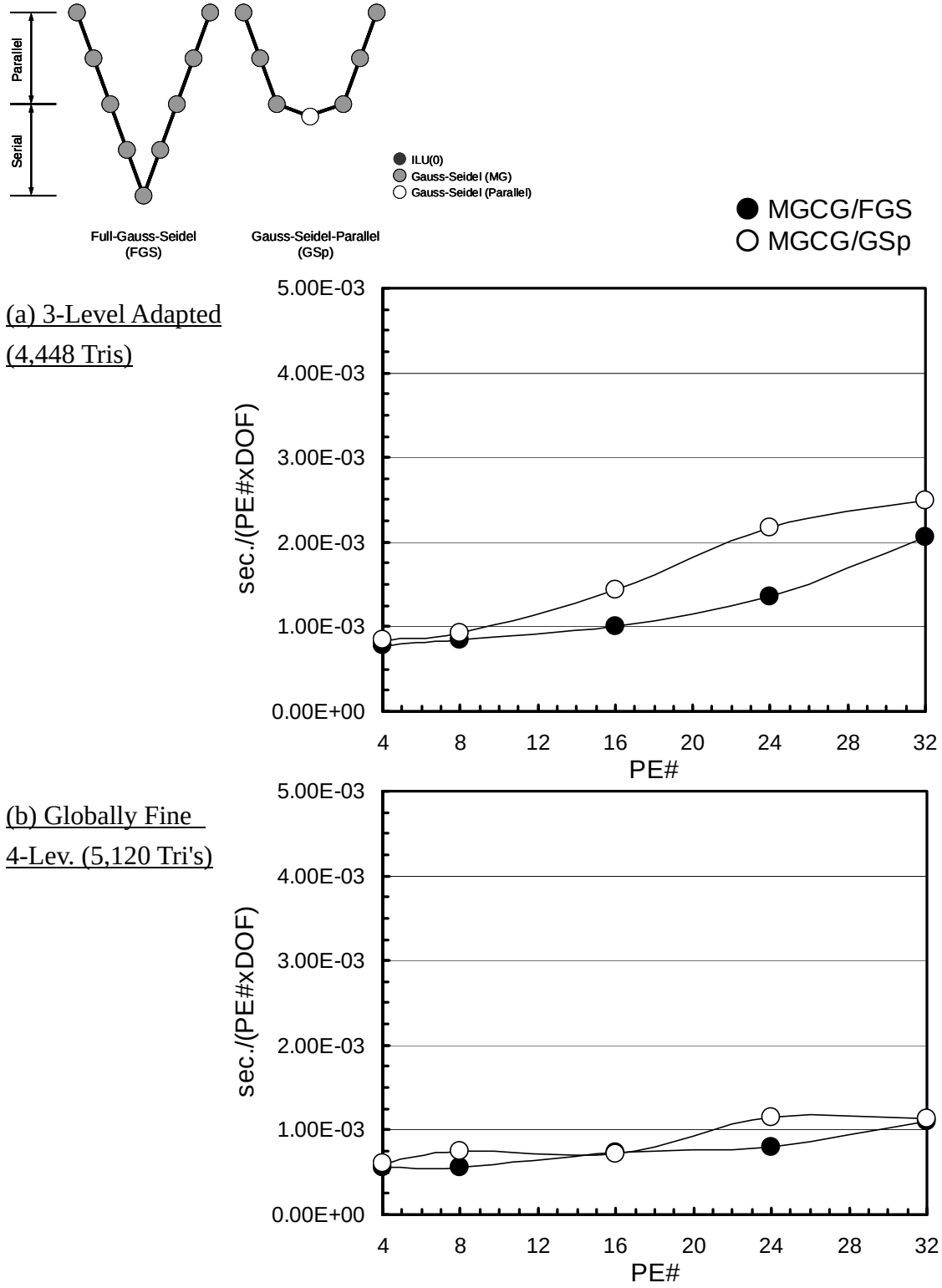


Fig. 5.42 Results of Poisson-II. Computation time (including communication, normalized by cell number/PE for parallel computing) for locally refined grids, 50 layers/PE, for 4 to 32 PEs (up to 7,116,800 cells). Single-Patch B.C., Direct Jump. (Black: MGCG/FGS, White: MGCG/GSp)

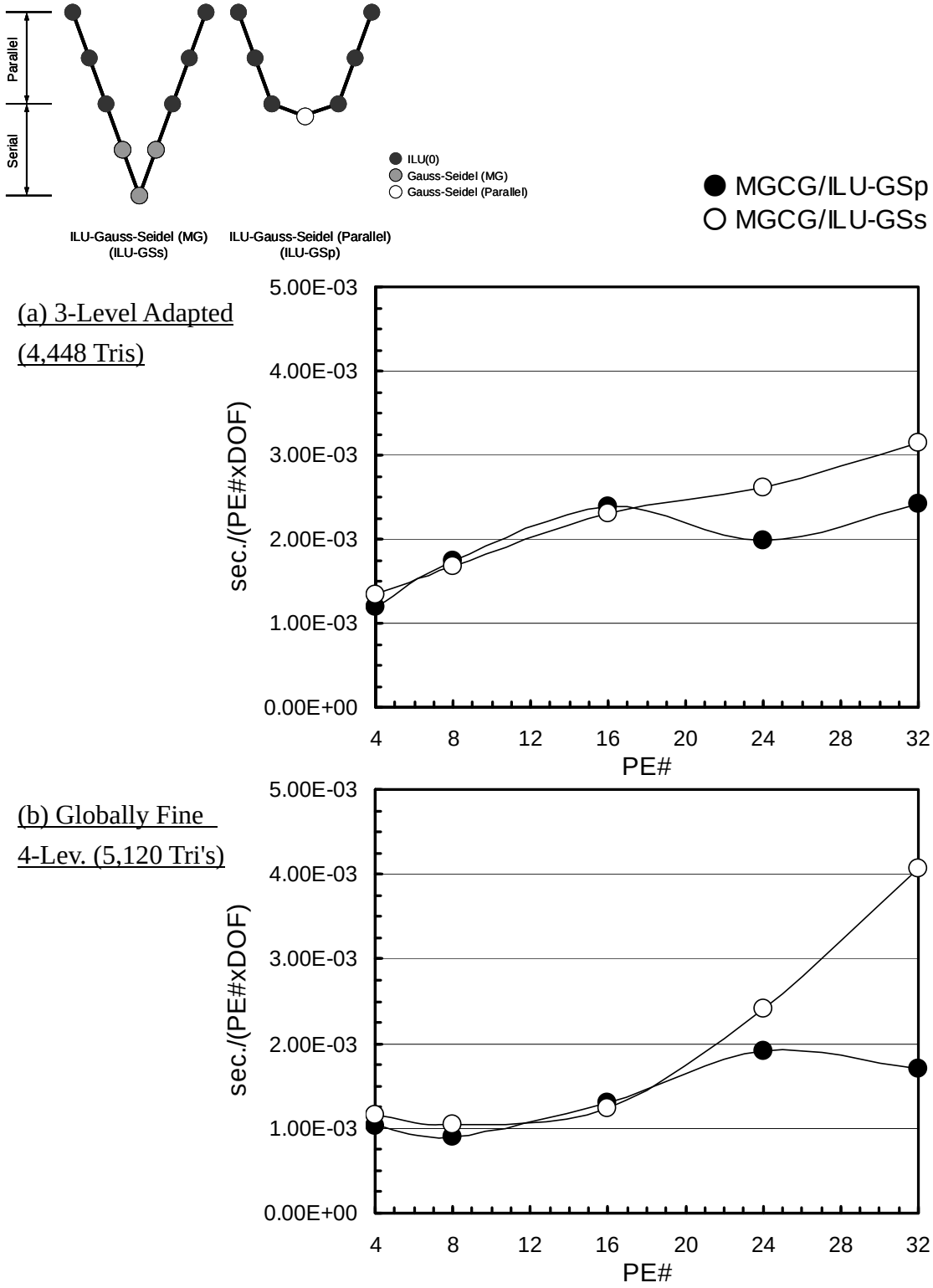


Fig. 5.43 Results of Poisson-II. Computation time (including communication, normalized by cell number/PE for parallel computing) for locally refined grids, 50 layers/PE, for 4 to 32 PEs (up to 7,116,800 cells). Single-Patch B.C., Direct Jump. (Black: MGCG/ILU-GSp, White: MGCG/ILU-GSs)

Table 5.1 Elapsed computation time including communication for 40 steps of thermal-convection simulations using 16 PEs of Hitachi SR2201. 5,120×25×16= 2,048,000 cells. Uniform B.C., $\Delta r = 1.25\times10^{-3}$.

	Momentum	Poisson+ Update	Energy
ICCG	176.6 sec.	4717.1 sec.	61.9 sec.
MGCG/FGS	176.6 sec.	3130.6 sec.	62.1 sec.
MGCG/ILU-GSp	176.6 sec.	3435.4 sec.	61.5 sec.