Efficient Computational Strategy for Ultra Large Scale Fluid Analysis

（超大規模流体解析の効率的計算手法）

Yasushi Nakabayashi

1

Efficient Computational Strategy for Ultra Large Scale Fluid Analysis

(超大規模流体解析の効率的計算手法)

Yasushi Nakabayashi

Acknowledgments

The author would like to express his sincere appreciation to Prof. G. Yagawa for his guidance and understanding during the course of this project. The author is pleased to acknowledge the considerable assistance of Profs. S. Yoshimura and N. Maeda. The author also would like to thank Prof. H. Okuda for suggesting this project and stimulating his interest.

The author is deeply indebted to Dr. Y. Wada and H. Takubo for their considerable assistance to this project, and to Dr. M. Shirazaki for his meaningful suggestion concerning CFD topics. The author wishes to express his appreciation to Prof. R. Shioya, Dr. H. Kawai, Dr. T. Miyamura, T. Yamada for their supports.

Many other people in the laboratory have also contributed to this study, Prof. T. Furukawa, H. Kawate, Dr. I. Shirai, Dr. T. Kowalczyk, A. Miyoshi, Y. Aoyama, K. Susa, S. Tanaka, K. Goto, T. Sugata, D. Ishihara and S. Nakagawa.

The author also would like to express his faithful gratitude to Prof. T. E. Tezduyar who gave him the opportunity to study at the University of Minnesota. Thanks also to all the friends, especially to N. Saeki and T. Yajima for their encouragement to carry out this project.

Finally, the author is deeply indebted to his parents, who gave him the opportunity to study at University of Tokyo for the completion of this project.

December, 1998

Yasushi Nakabayashi

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Finite Element Fluid Analysis

With the advancement of computer architecture, an object of computational fluid dynamics (CFD) is required to be large-scale, high-speed and high-accuracy. The finite difference method (FDM) has been mainstream of CFD, because of its simplicity of algorithm and low cost of CPU and memory storage. Otherwise, the finite element method (FEM), which has been used structural analysis, is attractive in terms of its applicabilities to an unstructured mesh and its simplicity for managing the boundary conditions. The bottle neck of FEM is large memory consumption and high cost of CPU.

The one-point quadrature (OPQ) technique proposed by Gresho et al. [1] is quite efficient in saving the computational storage. In this technique, which employs the Q_1-P_0 element; velocity-linear/pressure-piecewise constant, the diffusion and the advection matrices can be computed from the gradient matrix. Therefore the storage needed for the diffusion and advection matrices is unnecessary. Furthermore, using the matrix-storage free (MSF) formulation proposed by Okuda et al. [2], the gradient matrix can be computed from nodal informations i.e. nodal coordinates and element-node connectivities. Owing to the formulation, even the storage for the gradient matrix also becomes unnecessary. Furthermore, this approach is based on the element-by-element (EBE) scheme [3] [4] [5], which is the matrix solution algorithm without making global matrix. The EBE scheme with the conjugate-gradient (CG) type iterative solver is efficient for saving computational storage, since the storage becomes linear to the degree-of-freedom (DOF) of the analysis. Hence, a defect of FEM, large storage requirement, can be canceled.

Recent computer architecture is, however, apt to support large amount of memory system. Therefore MSF formulation is not always attractive fashion. Then the author compared three strategies of algorithm, OPQ, MSF and modified MSF [6]. OPQ is relatively high-accuracy and high-speed strategy, while MSF is small-memory and high-speed strategy. The modified MSF method is similar to the MSF method except for evaluation of advection term. Then the method is characterized to be high-accuracy and small-memory. It was observed that the EBE scheme is attractive for the parallel processing due to its locality of data structure [7] [8] The developed code was based on a EBE scheme, therefore parallel implementation was carried with no difficulty.

It is difficult for Galerkin FEM to solve imcompressible flow problems, especially in the case of high Reynolds number, without any stabilization method. One of the most famous approach for this problem is Streamline- Upwind/Petrov-Galerkin (SUPG) method [9], which uses modified interpolation function for the upwind scheme. Furthermore, Galerkin/Least-Square (GLS) method [10] and SUPG/PSPG (Pressure Stabilized Petrov Galerkin) method [11] are more robust method for stabilising both velocity field and pressure field. In this thesis, Balanced Tensor Diffusivity (BTD) method [1] was used for stabilizing technic. In the case of Q_1-P_0 element, BTD method is almost the same as SUPG method and spends less computational cost.

1.2 Parallel Processing

During the last several decades there has been an exponential growth in computing technology [12]. From 40s when the first developed computer, ENIAC appeared, microprocessors have speeded up 10 times in performance every 10 years. During the last decade they have doubled approximately every 18 months [13] and they continue to increase in performance.

Computer technology has solved or realized many difficult problems which had never been solved without a computer and still trying on more complicated problems. Today's advanced technology much depends on such a computing technology, therefore we can say that without it, many of advanced technology of nowadays can not be realized.

As the scale and complexity of target solved by a computer escalates, more computer power, i.e. calculation speed or memory size, are required. The more computer technology progress, the more they are used in various fields. Therefore more and rapidly progress is required for computer technology. They repeat themselves, that is, computer technology are fated to should be always in progress. To keep continuous progress and evolution in the future, it has been said that they have to break through technical and basic concept, that is, changing computing concept from a sequential computing to a parallel computing.

With a Neumann type computer, instructions and data streams are performed sequentially. Speeding up themselves was best way to develop a high performance computer, however, they encounter their physical limits, that is, they will never overcome a light speed. To overcome such a problem, we needed a new type of computing concept, i.e. parallel computing and a parallel computer. A parallel computer seemed to have infinite abilities and many parallel computers have been developed during last decades. Finally, a super parallel computer, i.e. massively parallel processors (MPP) which include thousands of processors have been developed and on the market.

On the other hand, in opposition to expensive super computers, economic workstations or more economic personal computers, in these years, we can not see so many differences between them, have spread and with a computer network which also have spread with astonishing speed, a virtual parallel computer, it is sometime called workstation clusters (WSC) and personal computer cluster (PCC), which means using workstations or personal computers connected through a computer network as a parallel computer, is being one of the most popular and easiest way to realize a parallel processing [14].

Thus today's parallel computers including virtual one have enough power to solve a large scale and complicated problem that was considered impossible in a few years ago. With these progress of hardware, software is also developing for a parallel computer to realize a high performance, however, it is always behind hardware. While many researches are being done in these years, more applications or technique for a parallel computer are needed in various fields to bring out the ability of a parallel computer. [15] [16]

In this thesis, considering such trends of computing technology and requirements of solving large scale and complicated problems, a parallel implementation of finite element fluids analysis code was proposed. At first, SIMD type parallelization was performed on the KSR-1 and CRAY T3D using parallel fortran compilers. It is only inserting compiler directives in the fortran source for parallel fortran to parallelize code. Secondly MIMD type parallelization was performed on the Hitachi SR2201 and DEC Alpha Cluster using Message Passing Interface (MPI). Using MPI library, the developed code became flexible and robust.

It is demonstrated that the present system can solve a three-dimensional incompressible viscous flow analysis of over ten million element, which corresponds to one hundred million degree of freedom (DOF) in a high parallel efficiency.

1.3 Large Scale Problem

According to the recent computer progress, requirements for solving large scale, over one million DOFs, fluid problem is increasing. Tezduyar et al. showed the results of several million DOFs problems [17] [18]. There are, however, many difficulties for solving more large problems.

At first, CPU cost becomes extreamly large. In this thesis, parallel processing is one of the most important topics for this reason. Secondly, memory cost also becomes very large. As shown above, MSF formulation and EBE scheme are good approach for minimizing memory cost. MSF and EBE are also good approach for reducing CPU cost because CPU cost is approximately proportional to DOFs squared (without these scheme DOFs cubed). Finally, input and output data size become giga bytes order (in the case of over ten million DOFs problems). Some operating system (OS) cannot treat over two giga bytes files. Then data decomposition is inevitable for such large scale problems. By decomposing data file, parallel I/O can be available. It is also neccessary for performing I/O in the reasonable time.

These difficulty are not only in the solving problems but also in the pre/post processing. In the pre processing, mesh generation and domain decomposition spend large CPU cost, memory cost and hard disk. In the post processing, data conversion and visualization spend the same resource. Therefore, parallelization, economizing memory cost, data split and parallel I/O are also neccessary for pre/post processing sub-systems.

1.4 Objectives

In this thesis, the development mothod of finite element fluids analysis system and its implementation on MPP, WSC and PCC was presented, the requirements of such studies was described above. Then, the objectives of this thesis was as follows:

[1] Finite Element Fluid Analysis System

Developing the finite element fluid analysis system based on the Matrix-Storage Free formulation, which economize the memory consumption of the computer system and suitable for large scale problems. Developing pre/post processing sub-sytem too. In the system, flexibility, stability and robustness are required.

[2] Parallel Implementation

Implementation of the developed system on MPP, WSC and PCC. Both SIMD type implementation and MIMD type implementation are performed. SIMD type implementaion is performed on the Kendall Square Research KSR-1 and CRAY T3D, and MIMD type implementation is performed on the Hitachi SR2201 and DEC Alpha Cluster. As the results of these implementation, high scalability, that is, its performance increase with the number of processors for a large scale and complicated problem, is required.

[3] Ultra Large Scale Problems

As the results of achieving above two objectives, Performing several tens of million DOFs problem in the reasonable time.

1.5 Outline of The Thesis

At first, Chapter 2 will give a general discussion on a finite element fluid analysis in two section. In the first section, the basic method of finite element formulation is presented. The strategies for large scale problem, for example matrix-storage free formulation, is presented in the second section.

Chapter 3 refers to parallel processing, which includes a method to analyze parallel performance, historical transition of supercomputer, massively parallel processors as a trend of supercomputer and parallel implementation on Kendall Square Research KSR1, CRAY T3D, Hitachi SR2201 and DEC Alpha Cluster.

In Chapter 4, the overview of the developed system is shown. The strategies for the ultra large scale problems are also summarized.

Chapter 5 presents some numerical results. After validation of accuracy, large scale problem and parallel efficiency are evaluated, practical problem, like a flow in a station subway, is presented.

Finally, Chapter 6 will summarize conclusions of the thesis.

Chapter 2

Finite Element Fluid Analysis

2.1 Finite Element Formulation

2.1.1 Governing Equations of Incompressible Viscous Flow

In this study, three-dimensional incompressibility viscous flow was considered. Governing equations for such a flow are the Navier-Stokes equations and incompressibility constraint equation as follows:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\nabla p + \frac{1}{Re}\nabla^2 \mathbf{u} \tag{2.1}$$

$$\nabla \cdot \mathbf{u} = 0 \tag{2.2}$$

where $\mathbf{u}$, p and Re are the velocity, the pressure divided by the fluid density and the Reynolds number, respectively. The initial/boundary-value problem is considered with the condition as follows:

$$\mathbf{u} = \mathbf{u}_0 \qquad \text{on } \Gamma_{\mathbf{u}} \tag{2.3}$$

$$\tau = \tau_0 \qquad \text{on } \Gamma_\tau \tag{2.4}$$

where $\mathbf{u}_0$ and τ_0 are the Dirichlet and Neumann boundary condition value, respectively, enforced on the corresponding boundary $\Gamma_{\mathbf{u}}$ and Γ_τ. Here, τ is defined as follows:

$$\tau = -p\mathbf{n} + \frac{1}{Re}(\mathbf{n} \cdot \nabla)\mathbf{u} \tag{2.5}$$

Given an initial velocity field, which satisfies Eq.(2.2) and appropriate boundary conditions for $\mathbf{u}$, Eqs.(2.1) and (2.2) can be solved, in principal, for $\mathbf{u}$ and p as functions of space and time.

2.1.2 Finite Element Formulation

The finite element spatial discretizations of Eqs.(2.1) and (2.2) are performed using the Galerkin method with the mixed interpolation formulation as follows:

$$\mathbf{u} = \sum_{n=1}^{N_{node}} \Phi_n \mathbf{u}_n \tag{2.6}$$

$$p = \sum_{e=1}^{N_{elem}} \Psi_e p_e \tag{2.7}$$

where Φ_n is a C^0 piecewise trilinear basis function with respect to node n. Ψ_e is a C^{-1} piecewise constant basis function (unity on element e and zero on all other elements). N_{node} and N_{elem} are the total number of the nodes and elements, respectively. The Galerkin method uses these basic function as the weight function of the weighted residual equatioons. Eq.(2.1) times Φ_n and Eq.(2.2) times Ψ_e integrated on Ω are shown as follows:

$$\begin{aligned} &\int_\Omega \Phi_n \frac{\partial \mathbf{u}}{\partial t} d\Omega + \int_\Omega \Phi_n (\mathbf{n} \cdot \nabla)\mathbf{n} d\Omega + \int_\Omega \Phi_n \nabla p d\Omega \\ &\quad - \frac{1}{Re}\int_\Omega \Phi_n \nabla^2 \mathbf{u} d\Omega + \int_{\Gamma_\tau} \Phi_n (\tau - \tau_0) d\Gamma = 0 \end{aligned} \tag{2.8}$$

$$\int_\Omega \Psi_e \nabla \cdot \mathbf{u} d\Omega = 0 \tag{2.9}$$

Integral of the third and forth term of Eq.(2.8) by parts become as:

$$\int_\Omega \Phi_n \nabla p d\Omega = \int_\Gamma \Phi_n p \mathbf{n} d\Gamma - \int_\Omega \nabla \Phi_n p d\Omega \tag{2.10}$$

$$-\frac{1}{Re}\int_\Omega \Phi_n \nabla^2 \mathbf{u} d\Omega = \frac{1}{Re}\int_\Omega (\nabla \Phi_n \cdot \nabla)\mathbf{u} d\Omega - \frac{1}{Re}\int_\Gamma \Phi_n (\mathbf{n} \cdot \nabla)\mathbf{u} d\Gamma \tag{2.11}$$

Here, Green-Gauss's theorem described below was adopted.

$$\int\int\int_v u \nabla w dv = \int\int_s (uw)\mathbf{n} ds - \int\int\int_v w \nabla u dv \tag{2.12}$$

The sum of the first term of the right-hand side of Eq.(2.10) and the second term of the right-hand side of Eq.(2.11) equal to τ in the fifth term of left-hand side of Eq.(2.8), then Eq.(2.8) can be rearranged to the weak-form as follow:

$$\begin{aligned} &\int_\Omega \Phi_n \frac{\partial \mathbf{u}}{\partial t} d\Omega + \int_\Omega \Phi_n (\mathbf{u} \cdot \nabla)\mathbf{u} d\Omega - \int_\Omega \nabla \Phi_n p d\Omega \\ &\quad + \frac{1}{Re}\int_\Omega (\nabla \Phi_n \cdot \nabla)\mathbf{u} d\Omega - \int_{\Gamma_\tau} \Phi_n \tau_0 d\Gamma = 0 \end{aligned} \tag{2.13}$$

Sunstituting Eqs.(2.6) and (2.7) into Eqs.(2.13) and (2.9) lead to the following integral equations.

$$\begin{aligned} &\int_\Omega \Phi_n \quad \Phi_m \quad \frac{\partial \mathbf{u}_m}{\partial t} d\Omega + \int_\Omega \Phi_n \Phi_l (\mathbf{u}_l \cdot \nabla \Phi_m)\mathbf{u}_m d\Omega - \int_\Omega \Psi_d p_d \nabla \Phi_n d\Omega \\ &\quad + \frac{1}{Re}\int_\Omega (\nabla \Phi_n \cdot \nabla \Phi_m)\mathbf{u}_m d\Omega - \int_{\Gamma_\tau} \Phi_n \tau_0 d\Gamma = 0 \end{aligned} \tag{2.14}$$

$$\int_\Omega \Psi_e \nabla\Phi_m \cdot \mathbf{u}_m d\Omega = 0 \tag{2.15}$$

Using the matrices, Eqs.(2.14) and (2.15) can be expressed as follows:

$$\mathbf{M}\dot{\mathbf{u}} + \mathbf{B}(\mathbf{u})\mathbf{u} - \mathbf{C}\mathbf{p} + \mathbf{D}\mathbf{u} = \mathbf{f} \tag{2.16}$$

$$\mathbf{C}^t\mathbf{u} = 0 \tag{2.17}$$

M, **B**, **C** and **D** are mass, advection, gradient and diffusion matrices, respectively. **f** is nodal external force vector. For the time marching method, Marker-and-Cell (MAC) algorithm with Adams-Bashforth (AB) time integration was employed. In this method, Eq.(2.16) and (2.17) are transformed into the explicit scheme of velocity field and the discretized Poisson equation for the pressure as follows:

$$\mathbf{u}_{n+1} = \mathbf{u}_n - \Delta t\mathbf{M}^{-1}(-\mathbf{C}\mathbf{p}_{n+1} + \mathbf{B}(\mathbf{u}_n)\mathbf{u}_n + \mathbf{D}\mathbf{u}_n - \mathbf{f}_n) \tag{2.18}$$

$$(\mathbf{C}^t\mathbf{M}^{-1}\mathbf{C})\mathbf{p}_{n+1} = -\frac{1}{\Delta t}\mathbf{C}^t\mathbf{u}_n + \mathbf{C}^t\mathbf{M}^{-1}(\mathbf{B}(\mathbf{u}_n)\mathbf{u}_n + \mathbf{D}\mathbf{u}_n - \mathbf{f}_n) \tag{2.19}$$

where Δt and subscript n denotes time increment and time step corresponding to the time $n\Delta t$. After Eq.(2.19) is solved using Conjugate Gradient (CG) method preconditioned by diagonal scaling, the velocity field of next time step is calculated by Eq.(2.18).

2.1.3 Stabilizing Method

Galerkin FEM with no stabilizing method is corresponds to center differencial in the FDM. Therefore, it has potential numerical instabilities especially in the case of advection dominated, high Reynolds number, flow. This instability occures in the velocity field.

There is another source of instability for FEM fluid analysis. It is due to using inappropriate combinations of interpolations functions of velocity and pressure fields. Because section is one of such combinations, instabilities apper in both velocity and pressure fields.

For vanishing or reducing these instabilities, there are some approaches , SUPG, SUPG/PSPG and GLS methods. In these methods, one additional term apperes in the finite element formulation of Navier-Stokes equations. The shape of these terms are described in Table (2.1). In this table $R(\mathbf{u}, p)$ is defined as below:

$$R(\mathbf{u}, p) = \frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} + \nabla p - \frac{1}{Re}\nabla^2 \mathbf{u} \tag{2.20}$$

There is another stabilizing method, BTD method, which is delived from Euler integration of time marching scheme. In this method, one additional term apper in the Navier-Stokes equations.

$$\nabla \cdot \frac{\Delta t}{2}\mathbf{u}\mathbf{u} \cdot \nabla \mathbf{u} \tag{2.21}$$

In the case of Q_1-P_0 element, BTD method is almost the same as SUPG method and spends less computational cost. PSPG term is equal to zero in the case of Q_1-P_0 element. GLS term is the most robust method, on the other hand, it spend large CPU cost because it has so many additional terms. Therefore, in this thesis, BTD method is mainly adopted.

Table 2.1: Additional term of SUPG, SUPG/PSPG and GLS methods

SUPG	$\tau_{SUPG}\mathbf{u} \cdot \nabla\phi R(\mathbf{u}, p)$
SUPG/PSPG	$(\tau_{SUPG}\mathbf{u} \cdot \nabla\phi + \tau_{PSPG}\nabla\psi)\, R(\mathbf{u}, p)$
GLS	$\tau_{GLS}R(\phi, \psi)R(\mathbf{u}, p)$

2.2 Strategies for Large Scale Problem

2.2.1 Element-by-Element Scheme

For the large scale problems, we first employ the EBE scheme in the stage of solving Eqs.(2.18) and (2.19). A principal virtue of this approach follows from the fact that element contributions are computed independently. By circumventing the matrix assembly and restructuring the CG algorithm at the element level, the total storage of memory can be economized and linear to the DOF. In this scheme, the matrix-vector product of CG algorithm is computed as follows:

$$\mathbf{A}\mathbf{p} = \sum_{e=1}^{N_{elem}} \mathbf{A}_e \mathbf{p}_e = \sum_{e=1}^{N_{elem}} \mathbf{q}_e = \mathbf{q} \tag{2.22}$$

where $\mathbf{A}$, $\mathbf{p}$ and $\mathbf{q}$ denote the global matrix, the global vectors and the result of matrix-vector product, respectively. Subscript e denotes the matrix or vector of the element e. After producting element-wise, only the element vector $\mathbf{q}_e$ was assembled to the global vector. Owing to this scheme, the solution of Eqs.(2.19) and (2.19) can be taken effectively in parallel, since the element-wise calculations can perform without remote memory access i.e. high data-locality can be established. (see next Chapter for parallel processing)

2.2.2 One-point Quadrature and Matrix-Storage Free Formulation

In the FEM formulation, the mass, the advection, the gradient and the diffusion matrices of Eqs.(2.18) and (2.19) are all integrated in element-wise.

The mass matrix is diagonally lumped, which is for explicit time marching of velocity field (see Eq.(2.18)), as follows:

$$M^e_{\alpha\beta} = \int_{\Omega^e} \Phi_\alpha \Phi_\beta d\Omega = \delta_{\alpha\beta} \int_{\Omega^e} \Phi_\alpha d\Omega \tag{2.23}$$

where subscripts α and β denote the number of node in element e, which varies from one to eight in hexahedral element. Ω^e denotes the volume of element e. By virtue of this lumping, the inverse of the mass matrix in Eq.(2.18) becomes no longer necessary, and the amounts of memory consumption can be economized.

The other matrices, the advection, the diffusion and the gradient matrix, otherwise, can not be lumped, an integration in the element is required. Contribution from the element e of the the advection, the diffusion and the gradient matrix are described as follows:

$$B^e_{\alpha\beta} = \int_{\Omega^e} \Phi_\alpha \overline{v_j} \frac{\partial \Phi_\beta}{\partial x_j} d\Omega \tag{2.24}$$

$$D^e_{\alpha\beta} = \frac{1}{Re} \int_{\Omega^e} \frac{\partial \Phi_\alpha}{\partial x_j} \frac{\partial \Phi_\beta}{\partial x_j} d\Omega \tag{2.25}$$

$$C^e_{\alpha i} = \int_{\Omega^e} \frac{\partial \Phi_\alpha}{\partial x_i} \Psi_e d\Omega = \int_{\Omega^e} \frac{\partial \Phi_\alpha}{\partial x_i} d\Omega \tag{2.26}$$

where $\overline{v_j}$ is the most recent value of the velocity. The subscripts i and j denote components of coordination. In the typical finite element formulation, Eqs.(2.24), (2.25) and (2.26) are integrated using 8-point Gauss's numerical integration. The integration, however, requires large amounts of CPU cost, then these matrices must be stored on the memory during solving a problem.

By using one-point quadrature technique proposed by Gresho [1], Eqs.(2.24) and (2.25) can be written as:

$$B^e_{\alpha\beta} = \frac{1}{8} \overline{v_j}(0) C^e_{\beta j} \tag{2.27}$$

$$D^e_{\alpha\beta} = \frac{1}{Re} \cdot \frac{C^e_{\alpha j} C^e_{\beta j}}{\Omega^e} \tag{2.28}$$

where $\overline{v_j}(0)$ is the velocity evaluated at the center of the element.

Eqs.(2.27) and (2.28) give significant meaning that **B** and **D** can be computed from **C**, therefore the storage needed for **B** and **D** can be released. Even in this situation, the storage of the gradient matrix occupies almost half of the total memory.

Regarding the gradient matrix, the Matrix-Storage Free formulation proposed by Okuda [2] is available. Applying the divergence theorem to Eq.(2.26) and using the one-point quadrature again, we get:

$$C^e_{\alpha j} = \int_{\Gamma^e} \Phi_\alpha n_i d\Gamma = \frac{1}{4}(\Gamma^{(a)} n_i^{(a)} + \Gamma^{(b)} n_i^{(b)} + \Gamma^{(c)} n_i^{(c)}) \tag{2.29}$$

where (a), (b) and (c) denote three surfaces of element e on which node α is included. $\Gamma^{(a)}$ and $n_i^{(a)}$ are the area and the outward unit vector of the surface (a).

An area vector of each surface, which is used to evaluate Eq.(2.29), can be computed by taking an outer product of vectors of element edges. When the element surface is a flat plane, the above algorithm bears the same result as the exact integration.

It is clear that all data for computing gradient matrix can be derived from nodal informations. Hence, even the storage for the gradient matrix also become unnecessary, that is, all matrix component need not be stored on the memory. We can calculate all matrix using MSF formulation whenever needed.

An example of estimation of memory requirement of the MSF formulation compared with that of standard FEM and the OPQ is shown on Tables 2.2, 2.3 and 2.4, which show a memory requirement of one million element problem. According to these tables, the advantage of the MSF is undoubtedly proved.

Table 2.2: Memory consumption of standard FEM for one million problem

	Node	Elem.	Total	Memory
Integer	0	8	8	32 Mbytes
Double	13	133	146	1168 Mbytes
Total	13	141	154	**1200** Mbytes

Table 2.3: Memory consumption of the OPQ for one million problem

	Node	Elem.	Total	Memory
Integer	0	8	8	32 Mbytes
Double	13	33	46	368 Mbytes
Total	13	41	54	**400** Mbytes

Table 2.4: Memory consumption of the MSF for one million problem

	Node	Elem.	Total	Memory
Integer	0	8	8	32 Mbytes
Double	13	9	22	176 Mbytes
Total	13	17	30	**208** Mbytes

2.2.3 Hourglass Control

The one-point quadrature technique might cause the hourglass mode, which is a physically meaningless oscillations corresponding to zero energy mode of the diffusion matrix. Gresho [1] have proposed the hourglass controller, then Okuda [2] customized it. Hourglass controller is dumping matrices, which would be added to the diffusion matrix evaluated by the one point quadrature. The coefficients of the dumping matrices are obtained by considering the difference between the diffusion matrices calculated by the exact integration and by the one point quadrature. Assuming that the element has a rectangular parallel-piped shape, both $\mathbf{D_{ex}}$ (exact value of $\mathbf{D}$) and $\mathbf{D_{1p}}$ (evaluated by the one-point quadrature) are easily computed. Subtracting $\mathbf{D_{1p}}$ from $\mathbf{D_{ex}}$, we can get the following dumping matrix:

$$
\begin{aligned}
&\frac{h_x h_y h_z}{144}\left\{\frac{1}{h_x^2}+\frac{1}{h_y^2}+\frac{1}{h_z^2}\right\}\{H_{3D}\}\{H_{3D}\}^t \\
&+\frac{h_x h_y h_z}{48}\left\{\frac{1}{h_x^2}+\frac{1}{h_y^2}\right\}\{H_{2D1}\}\{H_{2D1}\}^t \\
&+\frac{h_x h_y h_z}{48}\left\{\frac{1}{h_y^2}+\frac{1}{h_z^2}\right\}\{H_{2D2}\}\{H_{2D2}\}^t \\
&+\frac{h_x h_y h_z}{48}\left\{\frac{1}{h_z^2}+\frac{1}{h_x^2}\right\}\{H_{2D3}\}\{H_{2D3}\}^t
\end{aligned}
\tag{2.30}
$$

where hx, hy and hz are length of element edge in x-, y- and z-directions,respectively. H_{3D}, H_{2D1}, H_{2D2} and H_{2D3} are four hour glass mode as follows:

$$
\begin{aligned}
&\{H_{3D}\} = (-1,1,-1,1,1,-1,1,-1) \quad \{H_{2D1}\} = (1,-1,1,-1,1,-1,1,-1) \\
&\{H_{2D1}\} = (1,1,-1,-1,-1,-1,1,1) \quad \{H_{2D3}\} = (1,-1,-1,1,-1,1,1,-1)
\end{aligned}
\tag{2.31}
$$

Assumption behind the above derivation procedure implies that the hourglass controller of Eq.(2.30), which incorporates the length of element edge, is quite reasonable as far as the element is not severely distorted.

The sample of hourglass mode appeared in the velocity field is shown in Figure 2.1, and the same velocity field corrected by hourglass controller is shown in Figure 2.2.

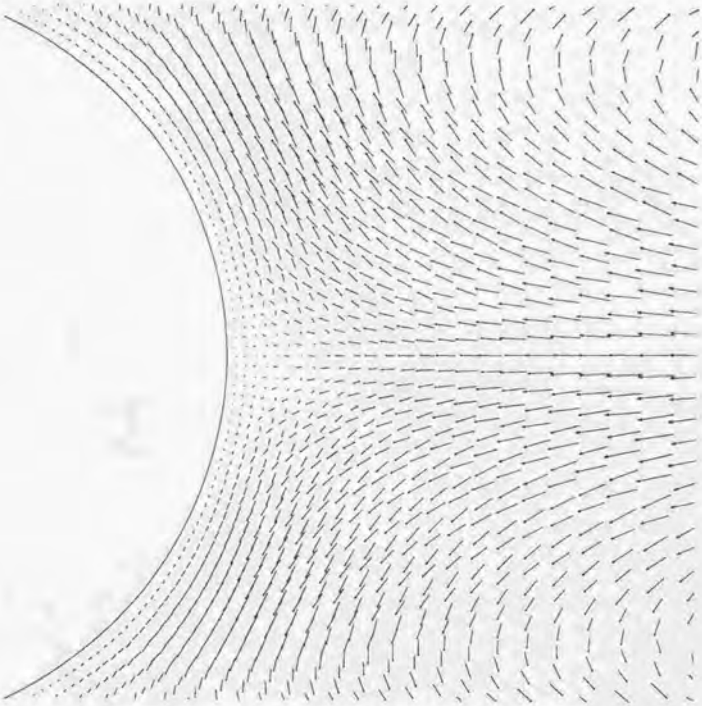


Figure 2.1: Velocity hog around circular cylinder with hourglass mode

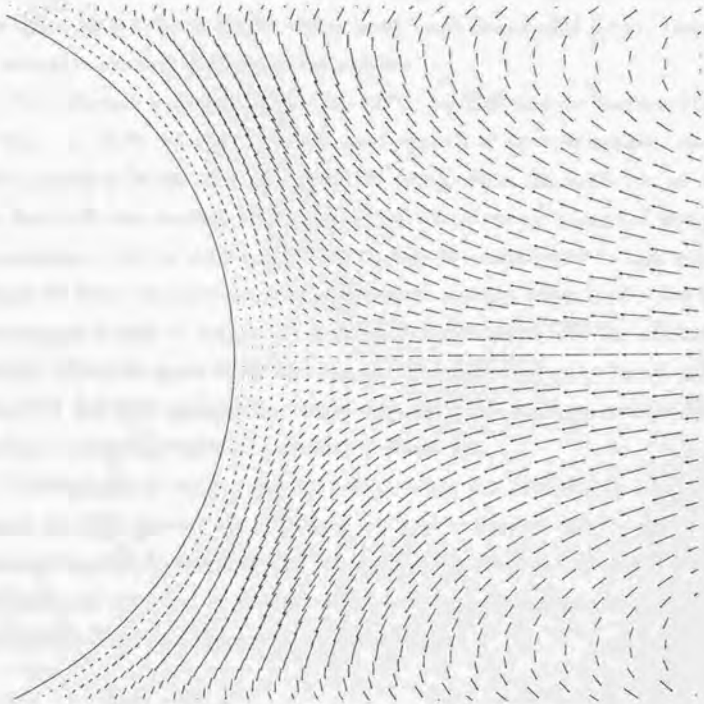


Figure 2.2: Velocity hog around circular cylinder without hourglass mode

2.2.4 Modified Matrix-Storage Free Formulation

Recent computer architecture is, however, apt to support large amount of memory system, MSF formulation is not always attractive fashion. Then the author consider a new strategy, i.e. modified MSF, which is the same as MSF except for the evaluation method of advection term. In this strategy, advection term is calculated by the Gauss quadrature using eight sample points every-time needed, for the purpose of saving the accuracy of advective eminent flow.

The difference of three strategies, the OPQ, the MSF and the modified MSF, is shown on Table 2.5 and 2.6, which are the result of three-dimensional cavity flow analysis of 32,768 elements and 35,937 nodes. From the Table 2.5, we can see that MSF and modified MSF strategies are advantageous in terms of memory consumption. On the other hand, OPQ strategy is attractive for its high speed. Figure 2.3 shows the convergence behavior of each strategy, which implies that the convergence of each strategy is not so different, but modified MSF has advantage slightly. Figure 2.4 shows the velocity of x-direction on vertical center line of cavity. The OPQ and MSF strategies has almost the same result, while the modified MSF strategy corresponds well to the results of Ku et al. [19]

Consequently, we can say that the OPQ strategy has advantage in computing speed, the MSF strategy has advantage in a high-balance of speed and memory consumption and the modified MSF strategy has advantage in both memory consumption and accuracy. A dynamic selection of these three strategies is required especially in large scale high performance computing.

Table 2.5: Comparison of memory consumption and CPU time

	OPQ	MSF	mod.MSF
Memory	14.6 M bytes	8.4 Mbytes	8.4 Mbytes
CPU time /step	9.5 sec	12.7 sec	28.6 sec
Formulation	(c),(d),(e)	(d),(e),(f)	(a),(e),(f)

Table 2.6: Formulation of advection, diffusion and gradient terms

$B^e_{\alpha\beta}$	$\int_{\Omega^e} \Phi_\alpha \overline{v_j} \frac{\partial \Phi_\beta}{\partial x_j} d\Omega$	(a)	$\frac{1}{8}\overline{v_j}(0) C^e_{\beta j}$	(d)
$D^e_{\alpha\beta}$	$\frac{1}{Re} \int_{\Omega^e} \frac{\partial \Phi_\alpha}{\partial x_j} \frac{\partial \Phi_\beta}{\partial x_j} d\Omega$	(b)	$\frac{1}{Re} \cdot \frac{C^e_{\alpha j} C^e_{\beta j}}{\Omega^e}$	(e)
$C^e_{\alpha i}$	$\int_{\Omega^e} \frac{\partial \Phi_\alpha}{\partial x_i} \Psi_e d\Omega$	(c)	$\frac{1}{4}(\Gamma^{(a)} n_i^{(a)} + \Gamma^{(b)} n_i^{(b)} + \Gamma^{(c)} n_i^{(c)})$	(f)

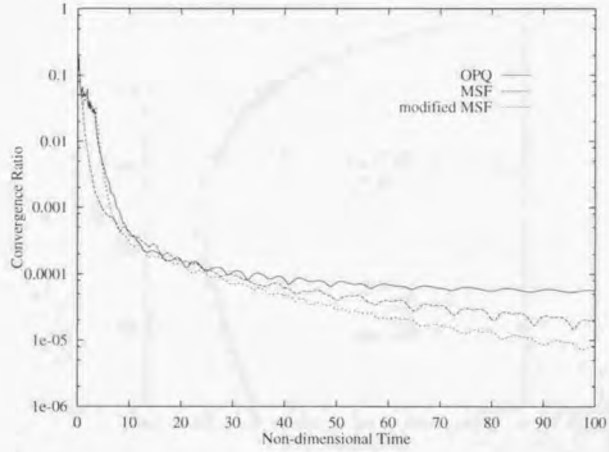


Figure 2.3: Convergence behaviors of three strategies

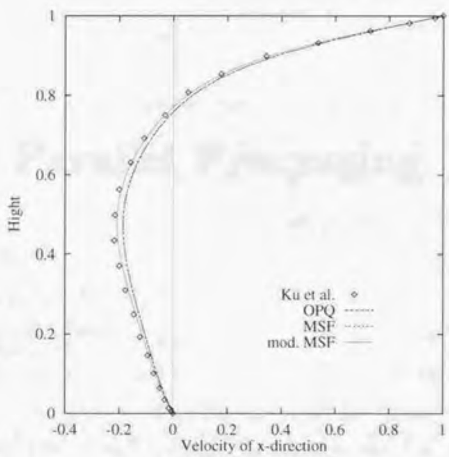


Figure 2.4: Velocity profiles of x-direction on vertical center line

Chapter 3

Parallel Processing

3.1 Parallel Processing

Parallel processing (computing) is defined as making some tools (processors) work for one target (for example, a numerical simulation). To complete a job which is over work for one person, there are two ways. First is making a improvement on a personal ability and second is making some persons work for the job, that is parallel processing. Since too many workers sometime cause low efficiency, it had been thought that using many processors can not be the best choice for solving a large scale problem which can not be solved by one processor. Nowadays, such a notion is denied and it is considered that arbitrary number of processors can be used in efficiency.

One of the most important requirements for parallel computing is achieving a high parallel performance. Generally, it depends much on an architecture of a parallel computer. It can be said for not only a parallel code, but also for any code which is specialized for speeding up that they should depend on computer's architecture to bring out computer characteristic. The more technique is used for a high performance, the more it should depend on an architecture. Even we can transplant an effective code which is developed on / for some computer considering its architecture to another computer which has different architecture, it is very difficult to keep a high performance. However, a supercomputer architecture is always changing in order to realize a high performance, therefore, it is an ideal for an effective code that can adapt to any kind of computer architectures.

In a parallel computer, since there is a variety of architectures, for example, network topology or memory distribution type, the above problem is more important between different type of parallel computers. In following sections, a general discussion on parallel computing and parallel algorithm which can achieve a high performance is described and recent parallel computers are reviewed.

3.1.1 Supercomputer

Computer engineering or science technology has been developing with high performance computers called a super computer. As shown in Figure 3.1, super computers have speeded up from 1 K (1.0×10^3) Flops (floating point operations per second) at 40s to just under 1 T (1.0×10^{12}) Flops at 90s. It seems that several computer architects will be able to break the T Flops peak performance barrier by the end of this century.

Table 3.1 shows historical change of cycle time of CRAY supercomputer which is typical of a supercomputer. Shown in this table, cycle time of microprocessor is rapidly speeding up, however, the progress of computing technology is not only from microprocessor's speed up but also from another technology's progress such as access speed to memory or hard disk, size of semiconductor memory chips or hard disk or communication speed between processors. These progress are summarized in [25] as follows:

- Since 1985 the performance of CMOS-based microprocessors has quadrupled every three years or at the rate of 60% every year. Clock speeds alone have evolved from 200 kHz in 70s to 300 MHz in 90s.

- Local area networks have improved by a factor of 10 every decade. In 80s ethernet operated at 10 M bits/sec. In 90s FDDI operated at 100 M bits/sec. Early prototypes indicate that G bits/sec networks will be commercially available by 2000.

- Computer backplane buses have improved by a factor of 10 every decade. Operating speed for buses have evolved from 2 M bits/sec in 70s to 640 M bits/sec in 90s.

- Semiconductor memory chips have quadrupled in capacity every three years since 70s. The number of bits per chip has increased from 1 K in 70s to 64 M in 90s.

Table 3.1: Cycle time of CRAY computers

Year	Model	cycle time [ns]
1976	CRAY 1	12.5
1982	CRAY X-MP	9.5
1985	CRAY 2	8.2
1988	CRAY Y-MP	6.5
1992	CRAY Y-MP C-90	4.0
1992	CRAY 3	2.0

- Magnetic disk storage has evolved from a density of 1 K bits per square inch in 50s to 1 G bits per square inch in 90s.

These combined performance growth have developed supercomputers like MPP.

Computer architecture is classified into simply four types [26], that is, SISD (Single Instruction-stream Single Data-stream), SIMD (Single Instruction-stream Multiple Data-stream), MISD (Multiple Instruction-stream Single Data-stream) and MIMD (Multiple Instruction-stream Multiple Data-stream) and MIMD is classified more detail [27] as shown in Figure 3.2. In [27], it is discussed that the candidate computer architectures and their potential to produce the T Flops systems. This figure includes not only MPP but also WSC.

As cost of workstations or personal computers down and also speed up, a virtual parallel computer using workstations or personal computers connected through a computer network, WSC, is being one of the most popular and easiest way to realize a parallel processing. In this section, MPP and WSC are described as a parallel computer and parallel programming systems are reviewed.

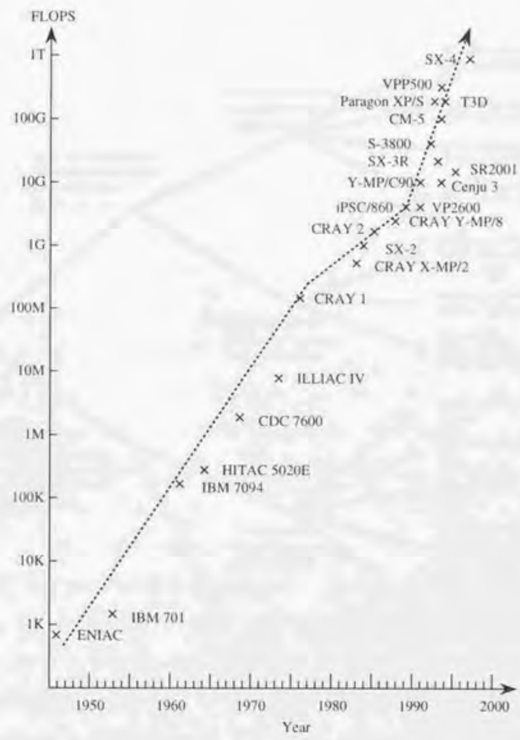


Figure 3.1: Peak performance vs development year of super computer

Figure 3.2: Taxonomy of tera computing

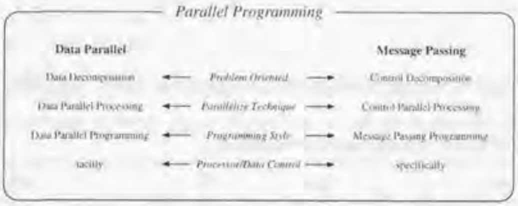


Figure 3.3: Data parallel and massage passing

3.1.2 Massively Parallel Processors

Table 3.2 [28] and 3.3 [29] shows peak performances of MPP at 1991 and 1995, respectively. These tables show what a fast speed MPP has been developing with.

Computer architecture of MPP is also classified into simply two types, SIMD and MIMD. Since SIMD is simpler architecture, early developed MPP (Active Memory Tech. DAP-610, MasPar MP-1 or Thinking Machines CM-2) adopted SIMD type and later, MIMD type MPP (Intel iPSC/860, NCube nCUBE2 or Fujitsu AP1000) have been developed. However, such MIMD type MPP can not support data parallel algorithm which is described in below.

To realize parallel programming on MPP, there are two types of programming. First is called "data parallel" programming and the other is called "message passing" programming. With the former, only data is parallelized and same instruction is performed to divided data in parallel and user have to consider only data dividing.

On the other hand, with the latter programming, not only data but also instructions are parallelized. It means that each processor has own instruction and data and to change data among processors, they communicate between processors specifically.

Figure 3.3 shows comparison between these parallel programming style.

Nowadays, MIMD type supporting both of parallel programming style, "Data Parallel" and "Message Passing" is being trend of MPP (Kendall Square Research KSR or Cray T3D).

Table 3.2: Peak performances of supercomputers in 1991

Model	Number of Processors	64-bit Gigaflops
Intel Delta	512	13.9
CRAY Y-MP C-90	16	13.7
NEC SX-3	2	10.0
Thinking Machines CM-200	2,048	9.8
Fujitsu AP-1000	512	2.2
nCUBE2	1,024	1.9

Table 3.3: Peak performances of supercomputers in 1995

Model	Number of Processors	64-bit Gigaflops
Intel Paragon XP/S MP	6,768	281.1
Numerical Wind Tunnel	140	170.4
Fujitsu VPP500/128	128	149.7
CRAY T3D 1024	1,024	100.5
Hitachi S-3000 cluster/412	12	78.2
Thinking Machines CM-5	1,024	59.7
IBM SP2-T2	256	44.2
NEC SX-3/44R	4	23.2

3.1.3 Speedup and Parallel Efficiency

To evaluate a parallel performance, "parallel efficiency" is used generally. Considering a sample case as shown in Figure 3.4, speedup with n processors, S_n is defined as follows:

$$S_n = \frac{t_1}{t_n} \tag{3.1}$$

where t_1 and t_n are total time for solving the problem using one processor and n processors, respectively. Using this S_n, parallel efficiency with n processors, E_n is defined as follows:

$$E_n = \frac{S_n}{n} = \frac{t_1}{n \cdot t_n} \tag{3.2}$$

According to Amdahl's law [20], these parameters have their limit. As shown in Figure 3.4, any task can be divided into parallelable portion $p(0 \leq p \leq 1)$ and non parallelable portion $(1-p)$. For parallelable portion, parallel computing with n processors can shorten calculation time to $1/n$ in ideal. However, for non parallelable portion, it has to been computed by only one processor. Then the whole computational time is evaluated as follows:

$$t_n = \frac{p \cdot t_1}{n} + (1-p)t_1 = (\tfrac{p}{n} + (1-p))t_1 \tag{3.3}$$

and speedup is:

$$S_n = \frac{1}{\frac{p}{n} + (1-p)} \tag{3.4}$$

Clearly, speedup S_n has following saturation point.

$$\lim_{n \to \infty} S_n = \frac{1}{1-p} \tag{3.5}$$

Equation (3.4) is sometime called Ware's law [21].

Table 3.4 shows variable speedup S_n to parallelable portion p and number of processors n. As shown in the table, it is difficult to keep a high parallel performance to increasing of the number of parallel processors. Amdahl's law for parallel processing which represents the limit of parallelism says that:

> Even when the fraction of serial work in a given problem is small, say $s(= 1-p)$, the maximum speedup obtainable from even an infinite number of parallel processors is only $1/s$.

In [22], it is said that to increase the number of processors, it is better to increase the scale of a problem. Generally speaking, parallel portion increase proportional to the scale of a problem and in an effective parallel algorithm, serial work portion should decrease to zero as the scale of a problem increase. In [23, 24], it is also shown that Amdahl's law is not important when parallelable portion increase proportional to the number of parallel processors.

As shown in Figure 3.5, when the problem is solved 1 time with n of processors and then sequential portion and parallelable portion are s or p, respectively ($s+p=1$), parallelable portion in whole computation increase proportional to the number of processors n and scaled speedup S_n is represented as:

$$\begin{aligned} S_n &= (s+np)/(s+p) \\ &= 1-p+np \\ &= n+(1-p)(1-n) \end{aligned} \tag{3.6}$$

Table 3.5 shows variable scaled speedup S_n to parallelable portion p and number of processors n. As shown in the table, scaled speedup is not their limit.

Table 3.4: Speedup with parallelable portion p and number of processors n

p	n=1	n=2	n=4	n=16	n=256	$n=\infty$
0.500	1.00	1.33	1.60	1.88	1.99	2.00
0.900	1.00	1.82	3.08	6.40	9.66	10.00
0.990	1.00	1.98	3.88	13.91	72.11	100.00
0.999	1.00	2.00	3.99	15.76	203.98	1000.00
1.000	1.00	2.00	4.00	16.00	256.00	∞

Table 3.5: Scaled speedup with parallelable portion p and number of processors n

p	n=1	n=2	n=4	n=16	n=256
0.500	1.00	1.50	2.50	8.50	128.50
0.900	1.00	1.90	3.70	14.50	230.50
0.990	1.00	2.00	3.97	15.85	253.45
0.999	1.00	2.00	4.00	15.99	255.75
1.000	1.00	2.00	4.00	16.00	256.00

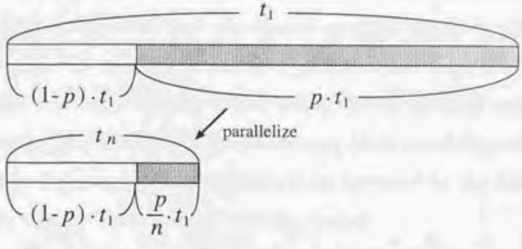


Figure 3.4: Parallelize portion

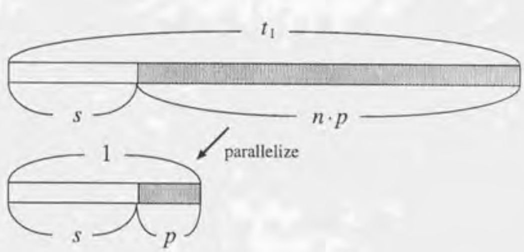


Figure 3.5: Scaled parallelize portion

3.2 SIMD-type Parallel Implementation

At first, a SIMD-type parallel implementation of finite element fluid analysis code is presented. The target machines of this implementation are Kendall Square Research KSR1 and Cray T3D, which have distributed memory system in physical point of view, while looks like the shared memory system in the users point of view. By virtue of such a memory system, called virtual shared memory, only MIMD-type (message passing) parallelization, but SIMD-type (data parallel) parallelization can be performed. Automatic data communication, operated by the hardware and/or the compiler, enables virtual shared memory system.

Data parallel style has advantage in the simplicity of programming and in the high load-balance. On the other hand, this style has a more communication overhead than message passing style, which is partly caused to the fact that the compiler for the parallel programs is not enough clever to optimize the parallel processing.

In this thesis, to realize large scale fluid analysis is one of the objectives. Therefore, a grain size of analysis decomposed into the number of processors is apt to be large. SIMD-type parallelization can be taken efficiently because of low portion of data communications compared with CPU cost in the each processors. Furthermore, EBE scheme, described in Section 2.2.1, is suitable for data parallel style. Since the program based on the EBE scheme is full of do-loops of total number of element, we must only decompose the do-loop to the number of processors we use.

3.2.1 Kendall Square Research KSR1

Kendall Square Research KSR1 [30] [31] is the first commercial MPP system which supports virtual shared memory system. The hardware overview of KSR1 is shown in Appendix A.

The method for parallelize the FORTRAN code is quite simple. Only inserting compiler directive in the FORTRAN source, we can parallelize the code. [32] [33] In practice, the process succeeds as follows:

1. Data distribution

   Data distribution to each processor's local memory has accomplished automatically by the hardware, named *ALLCACHE Engine*. Users need not be worried about it. *ALLCACHE Engine* optimizes the data distribution, furthermore, automatically date communication is performed by it.

2. Loop decomposition

   Loop decomposition to each processor can be performed the following compiler directive. In this case, the total number of element 'NE' is regularly distributed to the number of element divided by the number of processors.

```
c*ksr* tile (e)
      do e=1, NE
        A(e) = B(e)
      end do
c*ksr* end tile
```

In the programming environment of KSR1, automatically parallelization tool, named *KAP*, can be available, which analyzes FORTRAN source program and inserts compiler directives described above automatically. However, the performance of *KAP* is unsatisfactory, we should inset compiler directives carefully by hand.

3.2.2 CRAY T3D

CRAY T3D [34][35] is the same type of MPP system as the KSR1. T3D also support virtual shared memory system by software. The hardware overview of T3D is shown in Appendix B.

Parallelization mechanism is almost the same as that of KSR1 except for the date distribution declaration. The process of parallelization succeeds as follows:

1. Data distribution

   Data distribution to each processor's local memory has accomplished by the following simplest example of compiler directives. A shared array's dimension size must be declared to be a power of 2. If doing so makes an array elements can be left unused.

```
      dimension A(1024), B(1024)
cdir$ shared A(:BLOCK), B(:BLOCK)
```

2. Loop decomposition

   Loop decomposition to each processor can be performed the same sa KSR except for the directive's name and format, as follows:

```
cdir$ doshared (e) on A(e)
      do e=1,NE
        A(e) = B(e)
      enddo
```

3. Synchronization and serial region

   The all processor of T3D work in parallel all through the program without master directive, therefore explicit synchronization directive, barrier directive is necessary. The format of master directive and barrier directive are as follows:

```
cdir$ barrier
cdir$ master
      ....
cdir$ end master
```

3.2.3 Preprocessing for SIMD-type Parallel Processing

In this parallelization style (data parallel), the mesh data structure, which affect the performance of communication, is most important. For making data locality maximum, element and node renumbering is required. For example, when random numbering data is used, almost all the data is required to communicate over the remote processor's local memory.

In this study, the mesh generater for parallel processing is presented only for simple examples, square cavity and circular cylinder. The process of such a mesh generater is as follows:

1. Decomposition of physical analysis domain into the subdomains of the number of the processors. The process requires the optimization that the cross section of boundary between subdomains is minimumized in order to reduce date communications.

2. Giving global element and node number to each sundomain's elements and nodes in turn.

3. The optimized mesh data, which can be used as domain decomposision method with loop decomposision method, can be obtained.

When general unstructured mesh is supposed, any procedures such as the recursive spectral bisection (RSB) [36] and METIS [37] will be required.

3.3 MIMD-type Parallel Implementation

In this section, a MIMD-type parallel implementation of finite element fluid analysis code is presented. The target machines of this implementation are not restricted because Message Passing Interface (MPI) standard libraries absorb the difference of hardware architecture. Generally, using MPI (MIMD-type parallelization) result in high parallel efficiency because parallelable portion of code is increased and users can control more detail operations. On the other hand, implementation procedure is more difficult than SIMD-type implementation.

In this thesis, Hitachi SR2201 and DEC Alpha Cluster are used as the high-end machine of MPP and PCC, respectively.

The SR2201 employs an innovative three-dimensional crossbar switch to provide 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.

The DEC Alpha Clusters are connected via fast ether cable network, which is not fast compared with SR2201 network. However, using PC and ether network as the virtual parallel processors has the advantage in the point of economical view. In fact, recently, there are many WSC and PCC at the universities, laboratories and companies.

in the next sub-section basic method of using MPI libraly is shown, then the preprocessing method for MIMD-type implementation is shown the next sub-section.

3.3.1 Message Passing Interface (MPI)

Message Passing Interface (MPI) standard library is available on the both Fortran and C/C++ compilers. Here, basic and important functions of MPI are presented in the C form.

- Initialization

  At first, MPI_Init() function must be called for using MPI library. Then, MPI_Comm_size() and MPI_Comm_rank() give user the information of number of processes and process id. All the MPI program must finish MPI procedure with MPI_Finalize() function.

```
MPI_Init ( int *argc, char **argv[] );
MPI_Comm_size ( MPI_Comm comm, int *size );
MPI_Comm_rank ( MPI_Comm comm, int *rank );
MPI_Finalize (void);
```

- One-to-One Communication

  MPI_Send() and MPI_Recv() are the simplest functions of MPI, which are used for one-to-one communication. MPI_Send() function send the "count" of "dtype" data from buffer "*buf" to "dest" process. MPI_Recv() function receive the "count" of "dtype" data from "source" process into buffer "*buf". These function has some virsions; MPI_BSend(), MPI_SSend(), MPI_RSend(), MPI_ISend and so on. Fundamentaly, almost all the communication procedure can be described with only these two basic functions.

```
MPI_Send ( void          *buf    ,
           int            count  ,
           MPI_Datatype   dtype  ,
           int            dest   ,
           int            tag    ,
           MPI_Comm       comm     );
```

```
MPI_Recv ( void          *buf    ,
           int           count   ,
           MPI_Datatype  dtype   ,
           int           source  ,
           int           tag     ,
           MPI_Comm      comm    ,
           MPI_Status   *status    );
```

- Group Communication

MPI_Bcast() and MPI_Reduce() are the useful functions of MPI, which are used for group communication. MPI_Bcast() function send the "count" of "dtype" data from buffer "*buf" to all the process in the "comm" group. MPI_Reduce() function receive the "count" of "dtype" data from all the process in the "comm" group into buffer "*recvbuf" with the operation "op". MPI_Bcast() is useful function, for instace, one process input some data and send it to all the other processes. Furthermore, MPI_Reduce() is also useful, for example, in the case of computing the norm of some vectors.

```
MPI_Bcast ( void          *buf   ,
            int           count  ,
            MPI_Datatype  dtype  ,
            int           root   ,
            MPI_Comm      comm     );

MPI_Reduce ( void          *sendbuf ,
             void          *recvbuf ,
             int            count   ,
             MPI_Datatype   dtype   ,
             MPI_Op         op      ,
             int            root    ,
             MPI_Comm       comm      );
```

3.3.2 Preprocessing for MIMD-type Parallel Processing

In this parallelization style (message passing), two steps of pre-processing are required. They are mesh-generation and domain-decomposition. The first step, mesh generation, is almost the same as SIMD-type parallelization one except for node/element numbering. Node/element numbering in this step is not neccessary for MIDM-type parallelization because it is done in the next step. The second step, domain-decomposition, has many difficult requirement. At first, the cross section of domains should be minimum for decreasing the amount of data communication. Secondly, each domain has the same number of elements, node and boundary conditions for the load balance of each processor. Furthermore, these procedure should be in parallel because it needs large CPU and memory costs. The graph partitioning tool METIS/ParMETIS [37] [38] were adopted for the domain decomposition tool, which satisfied these requirement. The procedure of preprocessing can be summerized as follows:

1. Making the shape of analysis domain, then attach the boundary conditions.
2. Generating the elements/nodes using the ILA [39] or any commercial tool (e.g. KSWAD [40]).
3. Decomposition of domain by the METIS or parallel version of it, ParMETIS.

Chapter 4

System Overview

4.1 Finite Element Fluid Analysis System

The developed system is consist of three subsystems, which are preprocessing, fluid analysis and postprocessing subsystems. In each subsystem, the efficient method suitable for the large scale problem (c.f. Chapter 2 and 3) are adopted. Fig.(4.1) shows the whole system configuration, from which the flow of system becomes clear. In this section, the summary of each subsystem are shown.

4.1.1 Preprocessing Subsystem

There are three steps in the preprocessing subsystem, which are definition of the shape, mesh generation and domain decomposition. Each step has some option as below:

1. Definition of the shape
    - Commercial software (e.g. KSWAD)
    - Special tools for cavity and circular cylindar problems
    - By hand
2. Mesh generation
    - Commercial software (e.g. KSWAD)
    - Inteligent Local Aproach (ILA)
    - Special tools for cavity and circular cylindar problems
3. Domain Decomposition
    - METIS
    - ParMETIS

4.1.2 Fluid Analysis Subsystem

The key words of fluid anlysis subsystem are Matrix-Storage Free formulation and parallel processing. The system has many options for the flexibility and robustness. The system specification can be summarized as below:

1. Objects of analysis
   - Three dimensional flow
   - Incompressible viscous flow
   - Laminar flow
2. Method of analysis
   - Finite element method
     - Galerkin method
     - Q1-P0 hexahedral element
   - MAC method
     - CG solver
   - Time integration method
     - Euler method
     - Adams-Bashforth method
   - Matrix-Storage Free formulation
   - Balancing Tensor Diffusivity method
3. Parallel processing
   - SIMD type parallelization (parallel fortran)
   - MIMD type parallelization (MPI)
   - Parallel I/O

4.1.3 Postprocessing Subsystem

The aim of postprocessing is visualizing the result, for example, velocity field, pressure field and any other phisical values. Restart data file, which contain pressure value of each element and velocity value of each node, is available for the visualization of velocity and pressure fields. The fileter program can be used to convert restart data file into AVS UCD format file, which is available both AVS and parallel visualization system. The other phisical values, for example drag/lift coefficient, can be postprocessed using special tools.

1. Filter
    - Convert restart data file to the AVS UCD format file
    - Available for both AVS and parallel visualization tool
2. AVS
    - The most famous commercial software
3. Parallel Visulalization Tool
    - Visualizing pressure and velocity fields in parallel
    - Support AVS UCD format
4. Special Tools
    - Implanted in the mail system
    - Generate drag/lift coefficient profile

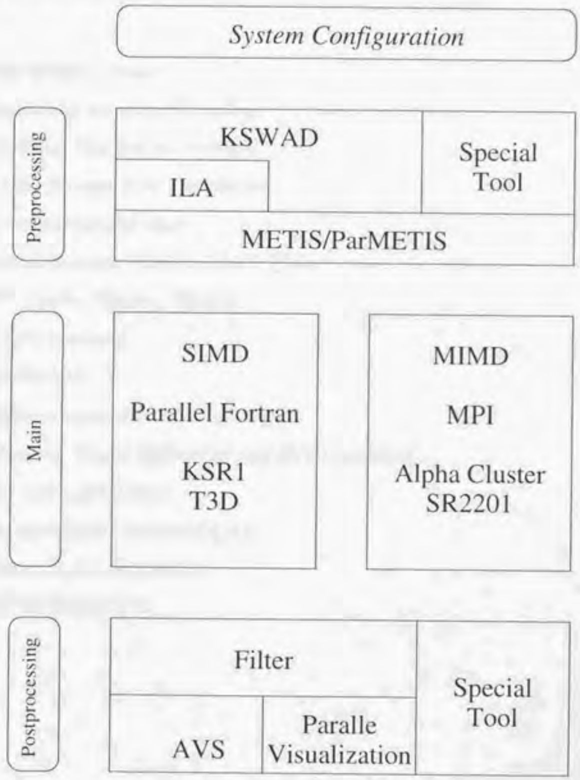


Figure 4.1: System Configuration

4.2 Strategies for the Ultra Large Scale Problem

In this section, the key technique for the ultra large scale problem are summarized. The detaile of these technique were described in chapter 2 and 3. Therefore, only the short lists are shown.

Minimizing memory usage

→ Element-by-Element CG solver

→ One-Point Quadrature method

→ Matrix-Storage Free formulation

Reducing computational time

→ Parallel Fortran (KSR1, CRAY T3D)

→ MPI (Alpha Cluster, SR2201)

Reducing I/O overhead

→ Parallel I/O

Stabilizing imcompressible fluid scheme

→ Balancing Tensor Diffusivity and SUPG method

Hexahedral mesh generation

→ Inteligent Local Aproach(ILA)

Efficient Domain decomposition

→ METIS/ParMETIS

Chapter 5

Numerical Results

5.1 Analysis Models

In this section, some numerical examples are shown. At first, flow around a circular cylinder is shown. The validation of accuracy of developed system is estimated in this analysis by compareing drag/lift coefficient. Next, three dimensional square cavity problem is shown. In this analysis, the validation of accuracy is done by checking x-direction velocity profile on perpendicular center line. This analysis is shown also in the later section of estimating parallel efficiency. Third analysis model is Yoshino river. This analysis is one of the practical problem using unstructured geometry and shows flexibility and robustness of developed system. Finally, Nezu model is shown. This analysis is also a practical problem and shows the feature of preprocessing subsystem.

- Flow around a Circular Cylinder
- Three Dimensional Cavity Flow
- Yoshino River
- Nezu Model

Table 5.1: Analysis conditions of cylinder flow

Cylinder diameter	1.0
Cylinder length	2.0
Domain size	20
Uniform velocity	1.0
Reynolds number	1000
Time increment	0.01

5.1.1 Flow around a Circuler Cylinder

As a sample problem to confirm the accuracy of the code based on the matrix-storage free formulation, a three-dimensional flow past a fixed cylinder at Reynolds number 1000 was computed. The analysis model is shown in Figure 5.1. The domain was decomposed to 80 elements around the cylinder, 40 elements in the radius direction and 10 elements in the axis direction; the total number of elements were 32,000. The element size near the cylinder surface is of the order 0.01. The analysis domain was set to 30 times of the cylinder diameter.

The upper and lower boundaries are flow symmetry lines, the upstream boundary is uniform velocity and the downstream boundary is traction-free. The analysis condition is shown in Table 5.1. A time step size of 0.01 was selected to provide sufficient stability in computation.

The simulation was continued for 8000 time steps, then reached a periodic state, as seen in the drag and lift coefficient plots in Figures 5.4, 5.5. The velocity hog and the pressure contour map of 8000 time steps was shown in Figures 5.2, 5.3. Although the oscillation in time space slightly araised from the effect of one-point quadrature and turbulence of flow, the result of this analysis almost coincides to the result by others [41] [42].

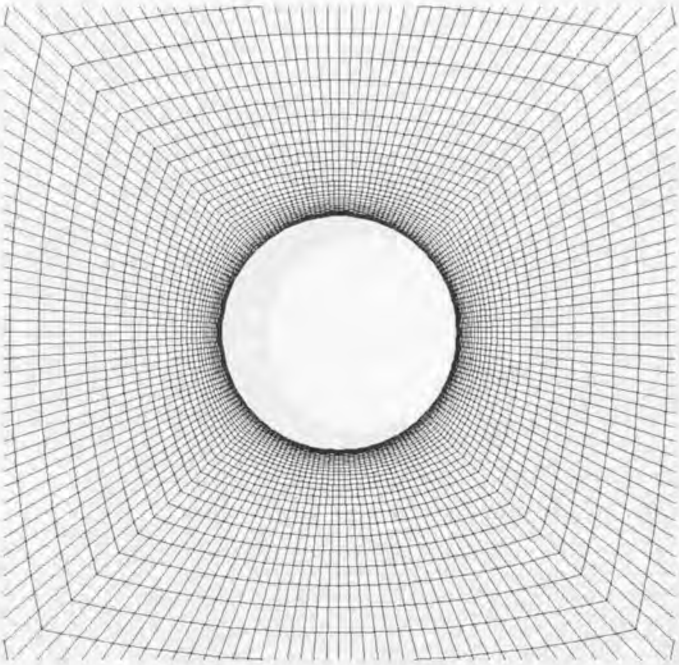


Figure 5.1: Mesh configuration of circular cylinder

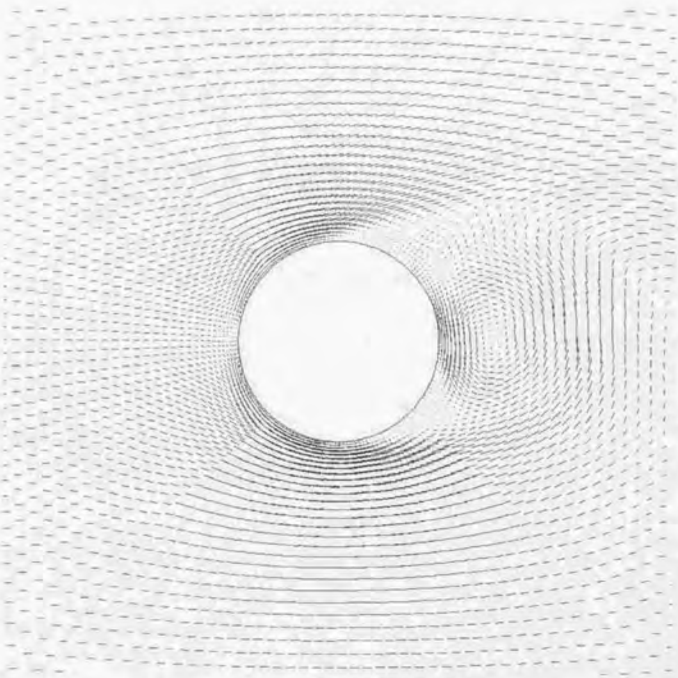


Figure 5.2: Velocity hogs around a circular cylinder at 8,000 time step

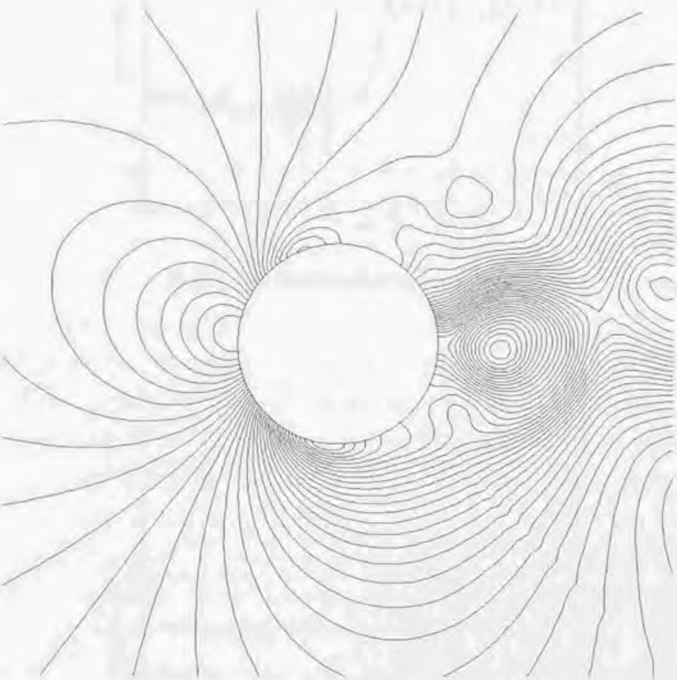


Figure 5.3: Pressure contour map around a circular cylinder at 8,000 time step

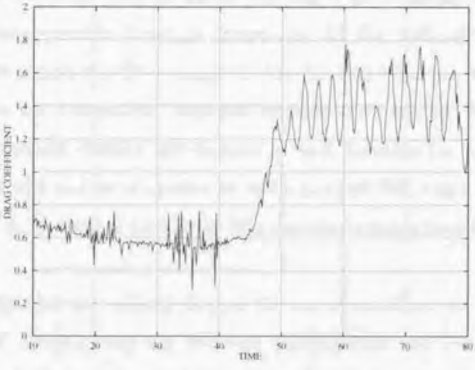


Figure 5.4: Profiles of drag coefficient

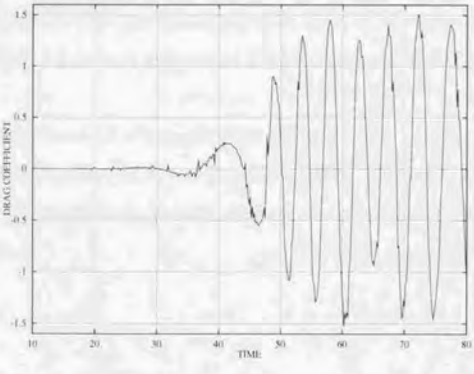


Figure 5.5: Profiles of lift coefficient

5.1.2 Three Dimensional Square Cavity Flow

For the example of large scale problem, we employ one million elements cavity problem. Analysis model is shown in Figure 5.6. All the walls of the cavity have no-slip boundary conditions (BC) except the top lid, which has the non-dimensional velocity of 1.0 in the x-direction. Reynolds number is set to 100. In this analysis, the cubic was regularly divided 100 element in each direction i.e. total number of elements is 1,000,000 and total number of nodes is 1,030,301, which corresponds to the total degree of freedom of 4,090,903. The non-dimensional time increment is set to 0.01.

Figure 5.7 and 5.8 are velocity hog at the converged state on plane α and β (see Figure 5.6), which shows the accurate velocity field and the appearance of three dimensional effect. Figure 5.9 show the velocity of x-direction on A-A' line (see Figure 5.6). The OPQ and MSF strategies has almost same result, while the modified MSF strategy correspond well to the results of Ku et al. [19]

In this analysis, the memory requirement is approximately 200 Mbytes. Therefore we can calculate this analysis even on the EWS (Sun SS-20) which has 256 Mbytes memory. The total analysis time needed for this analysis to converged is 216 hour, which correspond approximately nine days, in the case of EWS On the other hand, it needs 3.5 hour in the case of CRAY T3D used 128 CPUs (see Section 5.2.2).

Furthermore, as the result of using SR2201 the size of this analysis became 10 million elements, which correspond to the 40 million DOFs. Figures (5.10),(5.11),(5.12) shows the result of this analysis, parallel efficiency, convergence profile and pressure field, respectively.

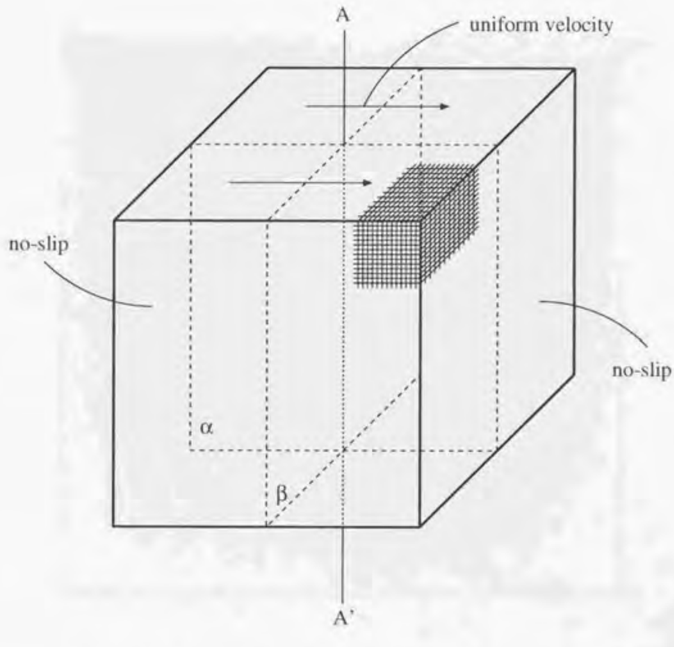


Figure 5.6: Analysis model of three-dimensional cavity flow

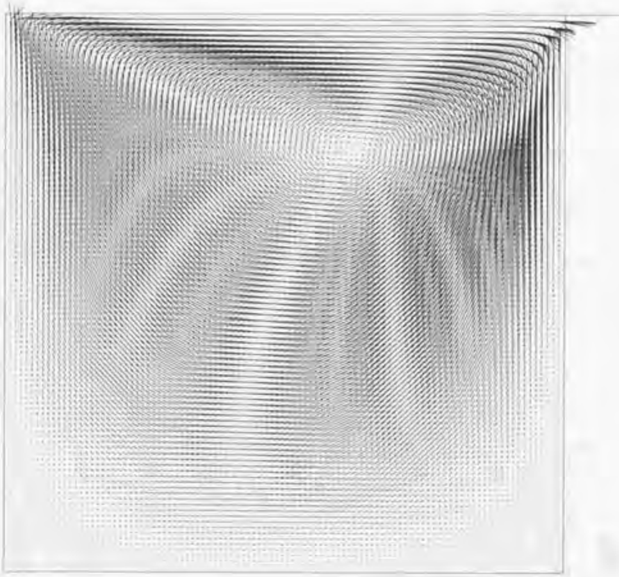


Figure 5.7: Velocity hogs of the converged state on the plane α

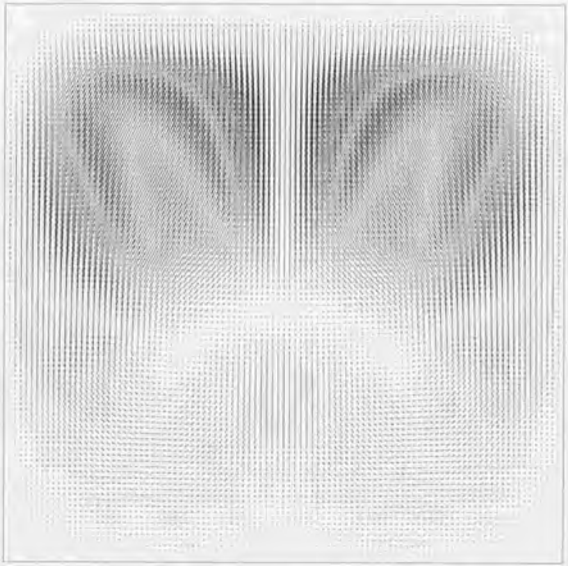


Figure 5.8: Velocity hogs of the converged state on the plane β

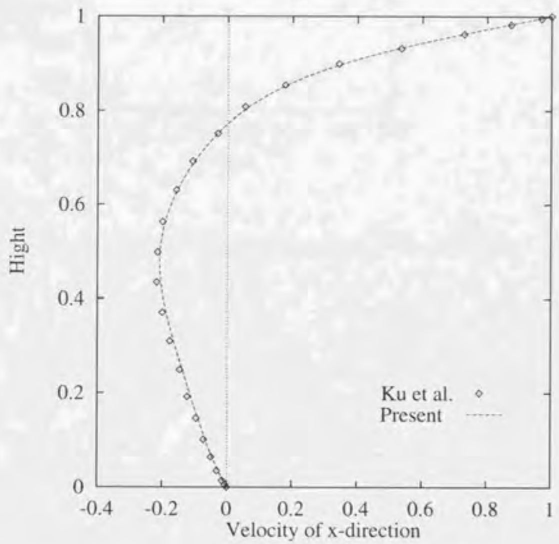


Figure 5.9: Velocity profiles of x-direction on A-A' line

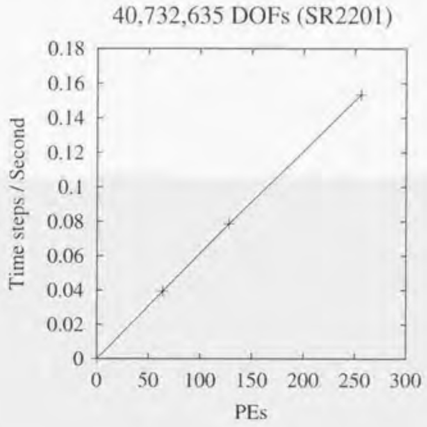


Figure 5.10: Parallel Efficiency of 40 Million Problem

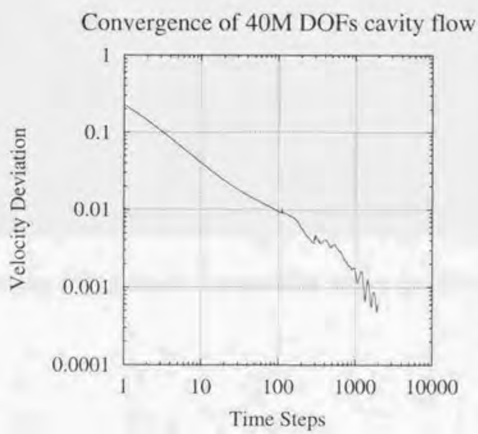


Figure 5.11: Convergence of 40 Million problem

Figure 5.12: Pressure Contour of 40 Million problem (1/8 model)

5.1.3 Yoshino River

For an example of practical problems, the flow of the Yoshino River was analized. In Figure 5.13, a model of the Yoshino River is shown, which is the area of eight kilometers form the mouth of the river.

The analysis domain was roughly decomposed into 4,000 element as two dimensional model, i.e. 200 elements in the direction of river flow and 20 element in the direction of vertical to river flow. The analysis condition is quite simple. All the boundaries are no-slip condition except for the mouth of the river and the upriver side. The mouse of the river, which is left-hand side in Figure 5.13, is traction free condition, and uniform velocity of 1.0 m/s from the upriver side, which is right-hand side in Figure 5.13, is given. The Reynolds number of this analysis and time increment are 1000 and 0.1, respectively.

After 5000 time steps of analysis, the velocity and pressure fields became steady states. The result of steady state in Figure 5.13 shows the pressure contour, which shows high pressure in red color and low pressure in blue color. Naturally, the mouse of the river show the most low pressure due to its traction free boundary condition. Otherwise, the most high pressure is shown in the projection of river, which is the junction of two river, because a flow of river hits against a river side in the point.

Althogh highly approximated analysis, the result of this analysis can be said reasonable.

Figure 5.13: Pressure contour of the Yoshino River

5.1.4 Nezu model

As a second example of practical problems, flow in a station of subway was analized. In Figure 5.14, a model of a station of subway is shown, which is simplified model of real station.

The analysis domain was decomposed into 15,840 elements and 20,886 nodes. The analysis condition of this model is that all the boundary are no-slip condition except for four square areas, i.e. two ends of a subway tunnel on lower-right side and left side, and two top stairs areas. The square area of lower-right side in Figure 5.14 has uniform velocity of 1.0 m/s. The square area of left-hand side and the two square areas of top stairs are traction free condition. The Reynolds number and time increment of this analysis are 1000 and 0.05, respectively.

It required approximately 4000 time steps for this analysis to be steady state. Then the velocity field of this analysis is shown using the tracks of particle tracers from the square area of lower-right side, which is shown in Figure 5.14.

In this analysis, the element data, i.e. coordinates of nodes and element-node connectivity, was made by ILA. After generating hexahedral element, the domain decomposition procedure was performed using ParMETIS. Fig.(5.15) shows the part of decomposed domain.

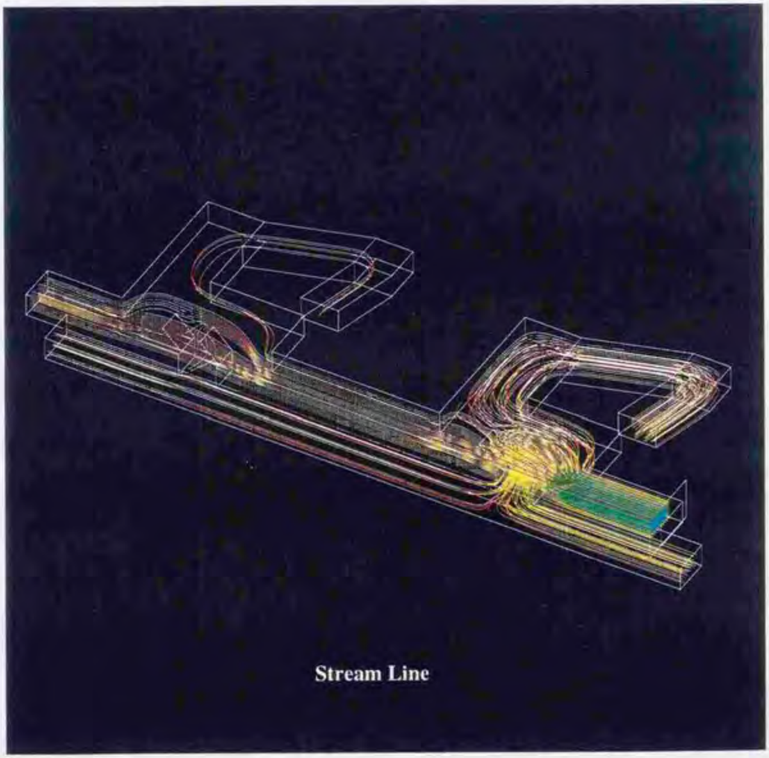


Figure 5.14: Tracks of particle tracers of a station of subway

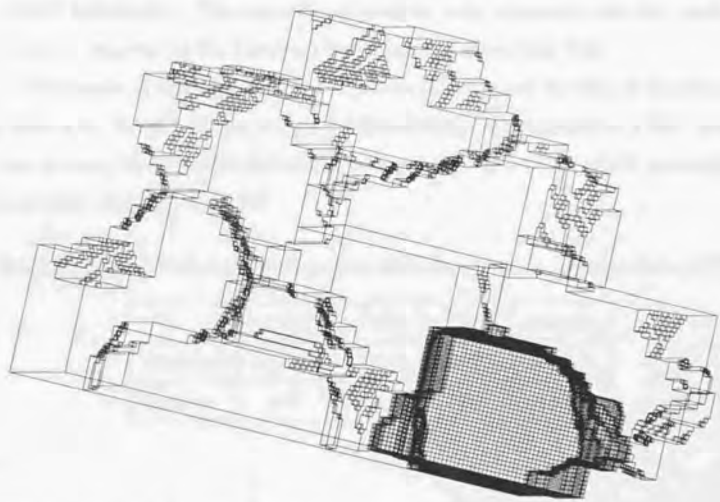


Figure 5.15: Decomposed element of Nezu model

5.2 Parallel Efficiency

5.2.1 Kendall Square Research KSR1

For a bench mark test to confirm parallel performance, the cavity flow, which is analysis almost the same as described in Section 5.1.2 except for the number of elements and nodes, are computed on KSR1 using 1,2,4,8 and 16 CPUs. In this analysis the number of elements are set to the two case of 16,384 (32x32x16) and 131,072 (63x63x32). The reduction of analysis scale compared with the previous section is required by the hardware limitation and accounting issue.

The result of this test is shown on Tables 5.2, 5.3 and the plot of speedup on Figure 5.16. We can see that a high parallel efficiency was attained by KSR1. In the case of using 16 CPUs and 131,072 elements, speed-up is 15.35, which corresponds to parallel efficiency of 95.9%.

Table 5.2: Parallel efficiency and speedup of 16,384 elements cavity analysis (KSR1)

	2 CPUs	4 CPUs	8 CPUs	16 CPUs
Efficiency	98.5%	97.78%	94.97%	90.73%
Speedup	1.97	3.91	7.60	14.52

Table 5.3: Parallel efficiency and speedup of 131,072 elements cavity analysis (KSR1)

	2 CPUs	4 CPUs	8 CPUs	16 CPUs
Efficiency	99.5%	99.3%	97.5%	95.9%
Speedup	1.99	3.97	7.80	15.35

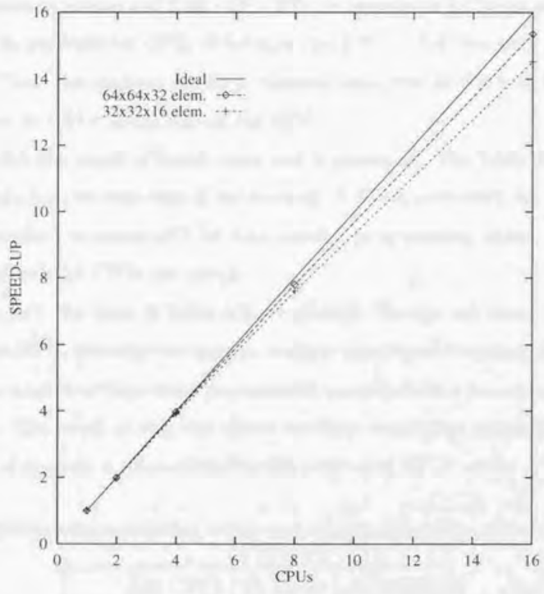


Figure 5.16: Speedup of cavity analysis compared with the ideal value (KSR1)

5.2.2 CRAY T3D

One million elements cavity analysis described in Section 5.1.2 is performed on CRAY T3D for evaluating parallel performance. The analysis model and condition are all the same as in Section 5.1.2, certainly, the analysis result is also the same as it.

In this bench mark test, not only the MSF but the OPQ are supposed because the total amount of memory of T3D (128 CPUs) is more than 2 Gbytes and enough memory can be available for OPQ. Otherwise, one CPU of T3D has only 16 Mbytes of memory. Then, this analysis can be performed using over 32 CPUs in the case of the MSF, over 64 CPUs in the case of the OPQ.

In Table 5.4, the result of bench mark test is presented. The table shows CPU time for calculating one time step of the analysis. T3D supports only the number of CPU corresponding to power of 2 for data parallel programming styles. Then, the case of 32, 64 and 128 CPUs are tested.

In Figure 5.17, the data of Table 5.4 are plotted. We can not know CPU time using only one CPU, therefore we can not evaluate speedup of this analysis. Instead of speedup, a number of time steps per second is presented for a measure of parallel performance. The result of this test shows excellent scalability, which means that the speedup of analysis is proportional to the number of CPUs used.

Table 5.4: CPU time for calculating a time step of one million elements cavity (T3D)

	32 CPUs	64 CPUs	128 CPUs
MSF	36.29 sec	18.72 sec	9.77 sec
OPQ	-	14.46 sec	7.62 sec

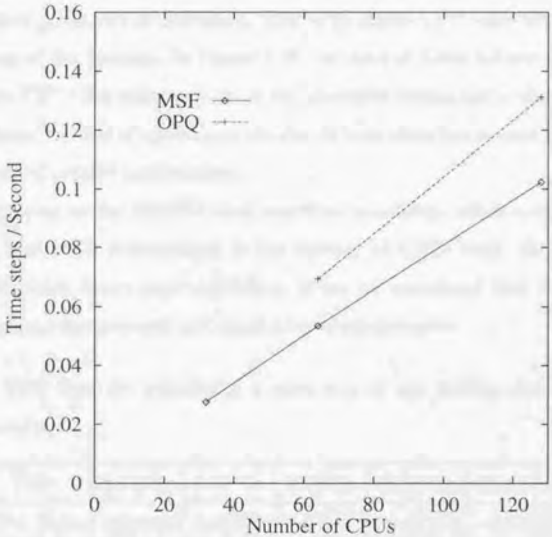


Figure 5.17: Parallel performance of one million elements cavity analysis (T3D)

5.2.3 DEC Alpha Cluster

The same problem of previous section, i.e. one million elements cavity analysis described in Section 5.1.2, is performed on DEC Alpha Cluster for evaluating parallel performance. The analysis model and condition are all the same as in Section 5.1.2, certainly, the analysis result is also the same as it.

In this test, the number of PEs are 2, 4, 8, 16 and 21. Each PEs has 512Mbytes memory and is connected 100Mbps fast ether network each other. In Table 5.5, the result of bench mark test is presented. The table shows CPU time for calculating one time step of the analysis. In Figure 5.18, the data of Table 5.5 are plotted. We can not know CPU time using only one CPU, therefore we can not evaluate speedup of this analysis. Instead of speedup, a number of time steps per second is presented for a measure of parallel performance.

The results up to the 16CPUs show excellent scalability, which means that the speedup of analysis is proportional to the number of CPUs used. In the case of 21CPUs, the result shows poor scalability. It can be considered that this result is caused by using ether network and conflicting communication.

Table 5.5: CPU time for calculating a time step of one million elements cavity (Alpha Cluster)

PEs	2 CPUs	4 CPUs	8 CPUs	16 CPUs	21 CPUs
CPU Time	0.00945	0.01877	0.03701	0.07292	0.08969

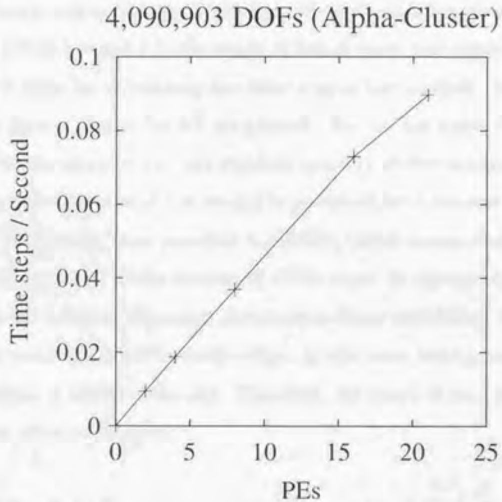


Figure 5.18: Parallel Efficiency of 1M Cavity on Alpha Cluster

5.2.4 Hitachi SR2201

The same problem of previous section, i.e. one million elements cavity analysis and 10 million element cavity analysis are performed on SR2201 for evaluating parallel performance. The analysis model and condition are all the same as in Section 5.1.2, certainly, the analysis result is also the same as it.

In this test, the number of PEs are 4, 8, 16, 32, 64, 128 and 256 for the 1M problem, and 64, 128 and 256 for the 10M problem. Each PEs has 1024Mbytes memory and is connected 300Mbps three dimensional crossbar network each other. In Tables 5.6 and 5.7, the result of bench mark test is presented. The table shows CPU time for calculating one time step of the analysis. In Figures 5.19 and 5.20, the data of Tables 5.6 5.7 are plotted. We can not know CPU time using only one CPU, therefore we can not evaluate speedup of this analysis. Instead of speedup, a number of time steps per second is presented for a measure of parallel performance.

The results show excellent scalability, which means that the speedup of analysis is proportional to the number of CPUs used. In the case of 25CPUs of one million element analysis, the result shows super-linear scalability. It can be considered that this result is caused by cache effect. In this case, total memory needed for the fulid analysis is within cache size. Therefore, the speed of each processor becomes higher than other conditions.

Table 5.6: CPU time for calculating a time step of one million elements cavity (SR2201)

PEs	8 CPUs	16 CPUs	32 CPUs	64CPUSs	128CPUs	256 CPUs
CPU Time	0.0499	0.0992	0.1941	0.3682	0.7166	1.4560

Table 5.7: CPU time for calculating a time step of ten million elements cavity (SR2201)

PEs	64CPUSs	128CPUs	256 CPUs
CPU Time	0.03938	0.07858	0.15340

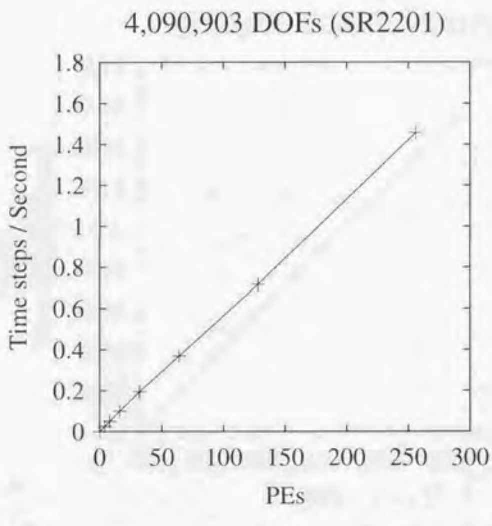


Figure 5.19: Parallel Efficiency of 1M Cavity on SR2201

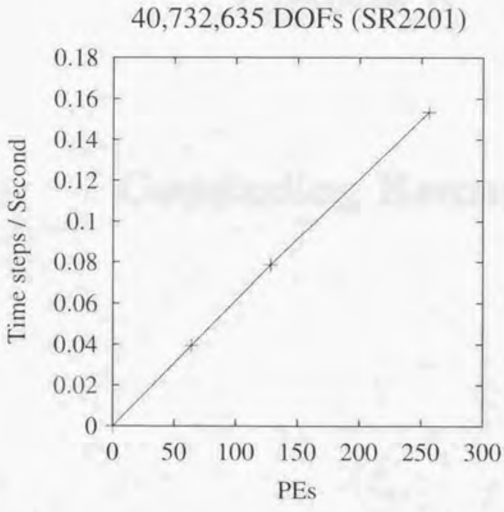


Figure 5.20: Parallel Efficiency of 10M Cavity on SR2201

Chapter 6

Concluding Remarks

- Finite element fluid analysis code, which is based on the Matrix-Storage Free formulations, was developed. Then, the code was implemented on the massively parallel processing computer system, Kendall Square Research KSR1 and CRAY T3D using compiler directives of parallel Fortran. Furthermore it was implemented on the SR2201 and Alpha Clusters using MPI library. In each case, high parallel efficiency was shown.

- Pre/post-processing sub-systems was also developed. Then the system was consist of modeling, mesh generation, domain decomposing, analysis and visualization sub-systems.

- As the ultra large scale problem, 40 million DOFs three-dimensional cavity problem was solved using 1024PEs of SR2201. The suitability for large scale problem of the developed code was shown by this analysis, and high parallel efficiency was also shown by the analysis.

Appendix A

Kendall Square Research KSR-1

KSR1 (Kendall Square Research) is the massively parallel computer systems, which can contain up to 1088 processors. In this study, we use KSR1-64, which has 64 processors. Each processor of KSR1 is a 64-bit processor, executes two integer and two floating instructions per cycle. The processor has the following registers and memory:

- 64 floating-point registers each, 64 bits wide.
- 32 integer registers, 64 bits wide.
- 32 addressing registers, each 40 bits wide.
- 0.5 MB of first-level cache, called the *subcache*, comprising 0.25 MB of instruction subcache and 0.25 MB of data subcache.

KSR1 has a memory system called ALLCACHE, which is distributed-shared memory as shown Fig.A.1. Each part of memory is distributed to each processor but the logical address space is continuous. Local cache is 32 MB which consists of some pages. A page is 16 KB , divided into 128 subpages of 128 bytes. Data is stored in units of pages and subpages and the unit of transfer and sharing between local caches is a subpage. Cache latencies is shown in Table A.1.

Table A.1: Cache Latencies of KSR1

Data Location	Cache capacity[MB]	Latency[clock]
Requestor's subcache	0.5	2
Local cache	32	20-24
ALLCACHE Group:0	1,024	175
ALLCACHE Group:1	34,816	600

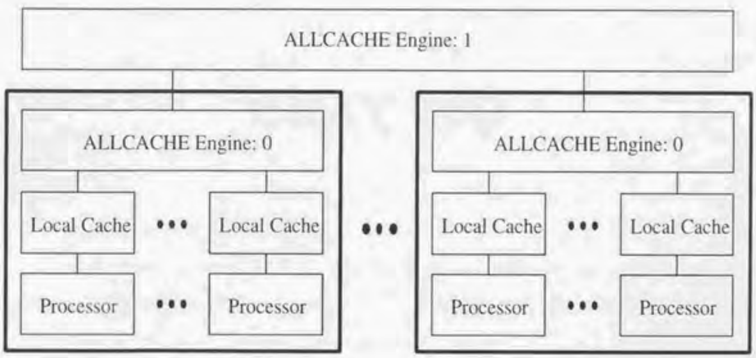


Figure A.1: ALLCACHE Engine

Appendix B

CRAY T3D

The CRAY T3D (MPP) system consists of processing elements (PE), each of which has a processor, associated logic and a connection to the interprocessor communication network. The processor is a DECchip 21064 microprocessor from Digital Equipment Corporation (DEC). The processor uses a Reduced Instruction Set Computing (RISC) architecture with a dual-issue, pipelined instruction stream.

A CRAY T3D system contains 32, 64, 128, 256, 512, 1024 or 2048 PEs available to users, depending on the system configuration. Each PE on the CRAY T3D system has three kinds of memory system that it can access:

- **Cache memory**

  Cache memory is a small, high-speed, random-access memory that temporarily stores frequently or recently accessed local data without user intervention. Data in cache memory can be accessed faster than data in local memory. It stores 1024 64-bit words of data as 256 cache lines. A cache line is 32 bytes or 4 words.

- **Local memory**

  Local memory consists of 64-bit words of dynamic random-access memory (DRAM). Each PE has its own local memory, and all PEs have the same amount of memory, either 2 Mwords or 8 Mwords.

- **Remote memory**

  Remote memory is all of the DRAM available on other PEs. All of the PEs can access the local memory of all other PEs, but the fastest access time always results from a PE accessing its own memory. Therefore, giving a PE work that involves accessing its own data will help maximize your program's performance.

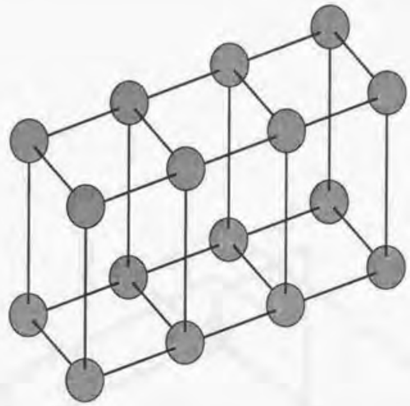


Figure B.1: A three-dimensional view of PEs

A CRAY T3D system also contains I/O gateway PEs that handle communication with the host system, either a CRAY C90 series system or other type of CRAY Y-MP system. The structure of the interconnecting network of PEs is three-dimensional in shape. Fig.B.1 illustrates the X, Y and Z axes of a three-dimensional partition of 16 PEs.

PEs on a CRAY T3D system can also communicate using communication paths linking PE nodes on opposite ends of a partition. These back-door paths are illustrated in the three-dimensional torus in Fig.B.2, which suggests the true physical layout of the CRAY T3D system.

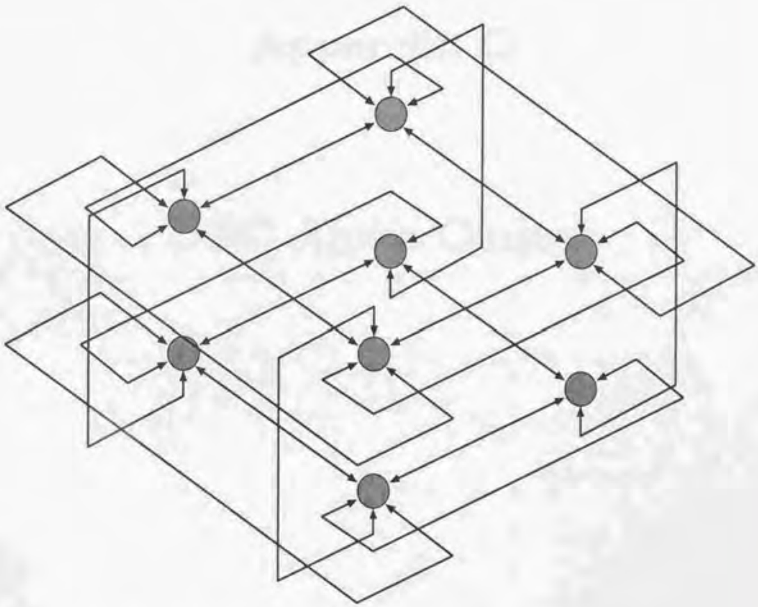


Figure B.2: Configuration of the three-dimensional torus network

Appendix C

DEC Alpha Cluster

DEC Alpha Cluster is original virtual parallel computer. The "virtual" means that it is really just a group of personal computers connected each other. Nevertheless, it shows the high-performance just like a parallel processors by virtue of the independence of the network from outer network (e.g. internet), high power CPUs, efficient file servers and so on.

It is consist of 24PEs of AlphaAXP 533MHz, 1PE of AlphaAXP 500MHz (front end machine) and 1PE of Sun UltraSPARC-I 200MHz. They are connected each other via 100Mbps fast ether cable network. Table (C.1) shows the detail hardware specification of Alpha Cluster and Fig. (C.1) shows the network configuration of whole Alpha Cluster system.

Table C.1: Alpha Cluster Hardware Specification

CPU	DEC AlphaAXP	DEC AlphaAXP	Sun UltraSPARC-I
Clock	500MHz	533MHz	200MHz
Memory	512MB	512, 1024MB	256MB
HardDisk	2G+2G	2G	4GB+50GB(RAID)
OS	Digital UNIX	Linux/Alpha	Solaris-2.6
Unit	1	16, 4, 4	1
Task	Compiling	Computation	File Server

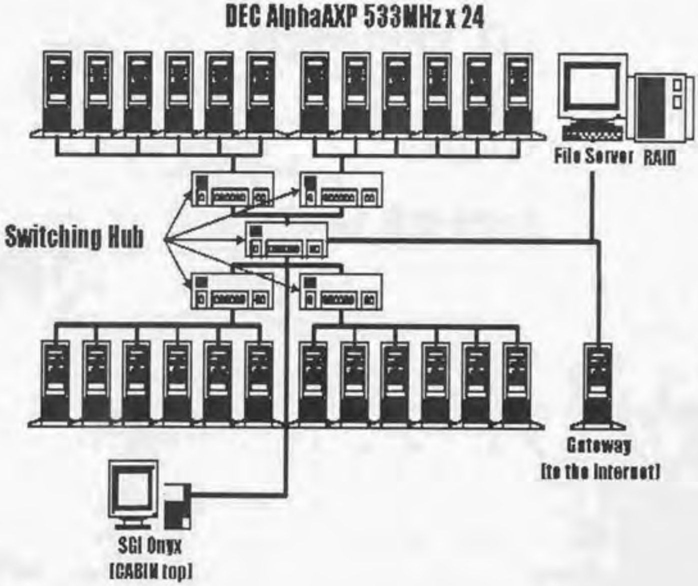


Figure C.1: Alpha Cluster network

Appendix D

Hitachi SR2201

The Hitachi SR2201 High-end model is capable of performing over 600 billion floating-point operations per second (600 GFLOPS;theoretical performance), and Compact model is capable of performing over 19 billion floating-point operations per second (19GFLOPS;theoretical performance). The SR2201 uses original RISC chips, and has a high level of scalability. The SR2201 High-end model ranges from 32 up to a maximum of 2,048 processors, and Compact model ranges from 8 up to a maximum of 64 processors. The SR2201 uses a crossbar switch network arrangement for high-speed message communication, and a pseudo vector processing function that provides a major boost in performance of large-scale scientific calculations.

- **Pseudo-vector processing function**

  The seudo-vector processing supports high-speed numerical calculations using multiple processing elements. Conventional RISC processors run into difficulty when data for large-scale numerical calculations cannot fit in their caches. The SR2201 series, however, employs pseudo-vector processing, a pipelining technique which fetches data(operands) directly from main memory (bypassing the cache) and into floating-point registers, without holding up the execution of subsequent instructions. This approach contributes significantly to high performance for large-scale numerical calculations.

- **Three-dimensional crossbar switch network**

  The High-end SR2201 employs an innovative three-dimensional crossbar switch to provide 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.

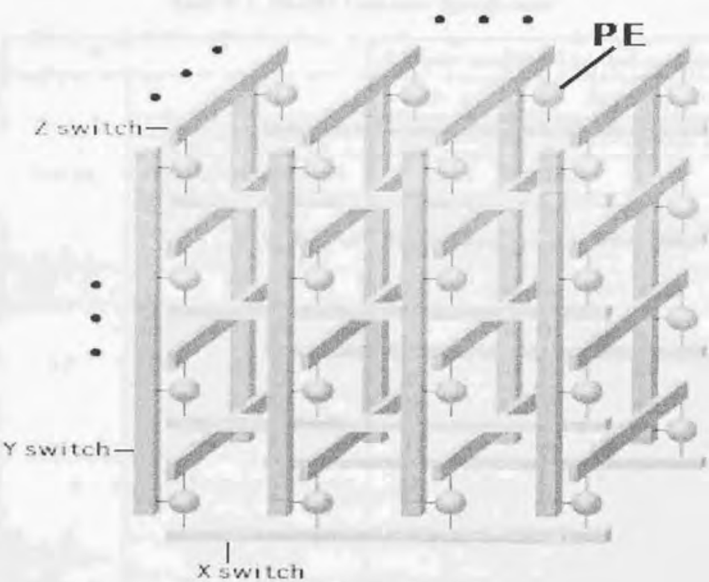


Figure D.1: Three-dimensional crossbar switch network

Table D.1: SR2201 Hardware Specification

		Compact models	High-end models
System	No.of PEs	8 - 64	32 - 2,048
	PE-PE network	1,2-D crossbar	2,3-D crossbar
	PE-PE transfer speed	300MB/s	
	Total storage	64GB	2TB
	External interfaces	Ethernet, Fast Ethernet, FDDI, HIPPI, ATM, Fast/Wide SCSI	
PE	Theoretical performance	0.3GFLOPS/PE	
	Addressable memory	64/128/256/512/1024 MB	
	Primary Cache	16kB(instruction), 16kB(data)	
	Secondary Cache	512kB(instruction), 512kB(data)	
I/O devices	Internal hard disk	4.3 GB/drive	2.1 GB/drive
	Disk array	4.6 - 66.6 GB/array (RAID5)	
	Internal DAT	2 - 8 GB(compressed)	
	External DAT	2 - 8 GB(compressed)	
	Console display	80 characters x 25 lines	

Bibliography

[1] P.M.Gresho, S.T.Chan, R.L.Lee and C.D.Upson, *Int. J. Num. Meth. Fluids* **4**, pp.557-598 (1984).

[2] H.Okuda and G.Yagawa, *Proc. of 2nd Japan-US Symposium on FEM in Large-Scale CFD*, pp.125-128 (1994).

[3] T.J.R.Hughes, M.Itzhak, I.Levit and J.Winget, Element-by-element solution algorithm for problems of structural and solid mechanics, *Comp. Meth. Appl. Mech. Engrg.* **36**, pp.241-254 (1983).

[4] G.F.Carey and B.N.Jiang, Element-by-element linear and nonlinear solution schemes, *J. Com. Appl. Num. Meth.* **2**, pp.145-153 (1986).

[5] L.J.Hayes and P.Devloo, A vectorized version of a sparse matrix-vector multiply, *Int. J. Num. Meth. Engrg.* **23**, pp.1043-1056 (1986).

[6] H.Okuda, Y.Nakabayashi and G.Yagawa, Control of accuracy and storage requirements of data-parallel finite element fluid analysis, *Proc. of 9th Computational Fluids Dynamics Symposium* pp.77-78 (1995).

[7] G.F.Carey, Parallelism in finite element modeling, *J. Com. Appl. Num. Meth.* **2**, pp.281-288 (1986).

[8] E. Barragy and G.F.Caray, A Parallel Element-by-Element Solution Scheme, *Int. J. Num. Meth. Engrg.* **26**, pp.2367-2382 (1988).

[9] A.N. Brooks and T.J.R. Hughes, Streamline Upwind/Petrov-Galerkin Formulations for Convection Dominated Flows with Particular Emphasis on the

Incompressible Navier-Stokes Equations, *Compt. Meth. Appl. Mech. Engrg.* **32**, pp.199-259 (1981).

[10] T.J.R. Hughes, L.P. Franca and G.M. Hulbert, A New Finite Element Formulation for Computational Fluid Dynamics: VIII. The Galerkin/Least Squares Method for Advection-Diffusion Equations, *Compt. Meth. Appl. Mech. Engrg.* **73**, pp.173-189 (1989).

[11] T.E. Tezduyar, Computation of compressible and incompressible flows with the clustered element-by-element method, *Univ. Minnesota, Supercomputer Institute Research Report*, **UMSI 90/215** (1988).

[12] G.S.Almasi and A.Gottlieb, Highly parallel computing, *The Benjamin / Cummings Publishing Company* (1994).

[13] BBN, Parallel computing past present and future, *Technical report, BBN Advanced Computers Inc., Cambridge, MA* **11** (1990).

[14] L.H.Turcotte, A survey of software environments for exploiting networked computing resources, *Engineering Research Center for Computational Field Simulation* (1993).

[15] R.Shioya, Massively parallel computing for large scale finite elements, *Ph.D thesis*, The University of Tokyo (1995).

[16] O.Aoki, A neural network-based finite element method on parallel processors, *Master's dissertation*, The University of Tokyo (1995).

[17] T. Tezduyar, S. Aliabadi, M. Behr, A. Johnson, V. Kalro and M. Litke, Flow simulation and High performance computing, *Comput. Mech.* **18**, pp.397-412 (1995).

[18] V. Kalro, S. Aliabadi, W. Garrard, T.E. Tezduyar, S. Mittal and K. Stein, Parallel finite element simulation of large ram-air parachutes. *Compt. Meth. Appl. Mech. Fluids*(1997).

[19] H.C.Ku, R.S.Hirsh and T.D.Taylor, *J. Compt. Phys.* **70**, pp.439-462 (1987).

[20] G.Amdahl, The validity of the single processor approach to achieving large scale computing capabilities, *Proc. The AFIPS Comp. Conf.* **30**, pp.483-485 (1967).

[21] W.Ware, The ultimate computer, *IEEE Spectrum* **3**, pp.89-91 (1973).

[22] C.Moler, A closer look at Amdahl's law, *Technical Report, Intel* **TN-02-687** (1987).

[23] J.L.Gustafson, Reevaluating Amdahl's Law, *Communications of the ACM* **31(5)**, pp.532-533 (1988).

[24] J.L.Gustafson, G.R.Montry and R.E.Benner, Development of parallel methods for a 1024-processor hypercube, *SIAM J. Sci. Stat. Comp.* **9**, pp.609-638 (1988).

[25] F.Cheong, OASIS: An agent-oriented programming language for heterogeneous distributed environment, *PhD thesis, The University of Michigan* (1992).

[26] M.Flynn, Very high speed computing systems, *Proc. IEEE* **54**, pp.1901-1909 (1966).

[27] G.Bell, Ultra computers: a Tera flop before its time, *Communications of the ACM* **35(8)**, pp.27-47 (1992).

[28] J.J.Dongarra, Performance of various computers using standard linear equations software, *Technical Report, Computer Science Department, University of Tennessee* **CS-89-85** (1991).

[29] J.J.Dongarra, Performance of various computers using standard linear equations software, *Technical Report, Computer Science Department, University of Tennessee* **CS-89-85** (1995).

[30] *KSR/Series Parallel Programming*, Kendall Square Research corporation, Mass. (1993).

[31] *KSR FORTRAN Programming*, Kendall Square Research corporation, Mass. (1993).

[32] H.Okuda, G.Yagawa and Y.Nakabayashi, Massively parallel fluid analysis by matrix-storage free finite element method, *Proc. of JSME 7th computational mechanics division*, (1994).

[33] H.Okuda, Y.Nakabayashi and G.Yagawa, Large scale finite element fluid analysis by massively parallel processors, *Proc. of 8th Computational Fluids Dynamics Symposium* pp.613-616 (1994).

[34] *Cray T3D System Architecture Overview* **HR-04033**, Cray Research, Inc. (1994).

[35] *Cray MPP Fortran Reference Manual* **SR-2504 6.2**, Cray Research, Inc. (1993).

[36] A.Pothen, H.D.Simon and K.P.Liou, Partitioning sparce matrices with eigenvectors of graphs, *SIAM J. Matrix Anal. Appl.* **11**, pp.430-452 (1990).

[37] G.Karypis and V.Kumar, A first and highly quality multilevel scheme for partitioning irregular graphs, *SIAM Journal on Sci. Compt.*, (1995).

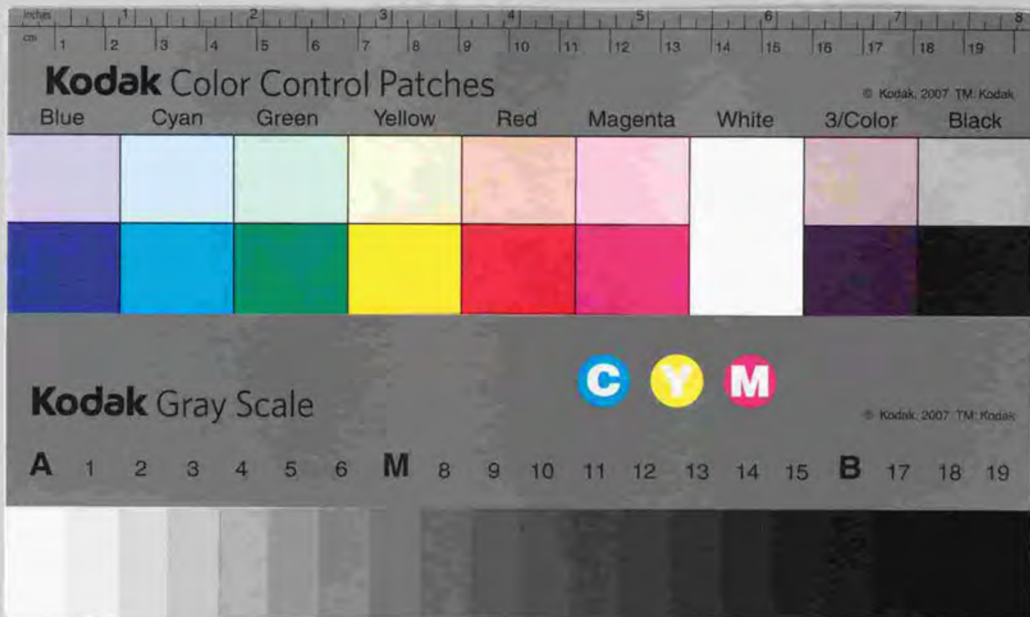
[38] G.Karypis and V.Kumar, Multilevel k-way partitioning scheme for irregular graphs, *Journal of Parallel and Distributed compting*, (1997).

[39] Y. Wada, *Ph.D thesis*, The University of Tokyo (1998).

[40] KSWAD manual,
(1998)

[41] M.Behr, A.Johnson, J.Kennedy, S.Mittal and T.Tezduyar, Computation of incompressible flows with implicit finite element implementations on the Connection Machine, *Comp. Meth. Appl. Mech. Engrg.* **108**, pp.99-118 (1993).

[42] H.Izumi, N.Taniguchi, Y.Kawata, T.Kobayashi and T.Adachi, Three-Dimensional Flow Analysis around a Circular Cylinder, *Trans. JSME Ser.B* **60**, pp.3797-3804 (1994).


Kodak Color Control Patches
© Kodak, 2007 TM: Kodak
Blue
Cyan
Green
Yellow
Red
Magenta
White
3/Color
Black
C Y M
Kodak Gray Scale
© Kodak, 2007 TM: Kodak
A 1 2 3 4 5 6 M 8 9 10 11 12 13 14 15 B 17 18 19