数理解析研究所講究録
989 巻 1997 年 132-147

Large Scale Finite Element Analysis on a Massively Parallel Computer

R. Shioya and G. Yagawa

School of Engineering, The University of Tokyo
7-3-1 Hongo Bunkyo-ku Tokyo 113, JAPAN
Tel: +81-3-3812-2111 ex. 6994 Fax: +81-3-5684-3265
E-mail: shioya@gen.u-tokyo.ac.jp

Abstract

This paper describes the parallel finite elements for MIMD type massively parallel computers and clustered workstations. As a parallel numerical algorithm for the finite element analyses, the present authors have utilized the Domain Decomposition Method (DDM) combined with an iterative solver, i.e. the Conjugate Gradient (CG) method where a whole analysis domain is fictitiously divided into a number of subdomains without overlapping. In order to solve the issue of memory shortage, the present system has adopted a hierarchical distributed data management system. The present method is successfully applied to over one million d.o.f. scale 3-D structural problems with high parallel efficiency of more than 90% with the 1,024 CPUs.

KEY WORDS: Parallel Finite Element Method, Domain Decomposition Method, Massively Parallel Processors.

1 Introduction

During the last several decades there has been an exponential growth in computing technology [1]. From 40s when the first developed computer, ENIAC, appeared, microprocessors have speeded up 10 times in performance every ten years. During the last decade they have doubled approximately every 18 months [2] and they continue to increase in performance.

With such a progress of computing technology, a numerical simulation like a finite element method (FEM), has been established as the third method followed by theory and experimentation. Consequently, numerical simulations are replacing experimental studies in fields where it takes enormous cost or time, or it is even impossible to carry out an experiment. In such fields, computer technology itself have been a target for study or research and, because of it, new fields of study have come out, being termed computer engineering, science and so on.

Computer technologies have solved many difficult problems which had never been solved without a computer, and are still trying on more complicated problems. The fact that today's technologies significantly depends on computer hardware indicates that they could not have been realized without computer developments.

As the scale and complexity of interest solved by a computer escalates, more computer power, i.e. speed or memory size, are required. The more the computer technology progresses, the more then they are used, thereby requiring more rapid progress. 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 concepts, that is, changing the computing concept from sequential computing to parallel computing.

With a Neumann type computer, instructions and data streams are performed sequentially. Speeding up themselves was the best way to develop a high performance computer, but, they encounter their physical limits, that is, they will never exceed 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 the last decades. Finally, a super parallel computer, i.e. massively parallel processors (MPP), which include thousands of processors, have been developed and appeared on the market.

On the other hand, in opposition to expensive super computers, more economical computers such as workstations or personal computers, which we cannot classify these days have spread, and with a computer network which also have spread with astonishing speed a virtual parallel computer, sometimes called workstation clusters (WSC), which are a set of workstations or personal computers connected through a computer network as a parallel computer, is being one of the most popular and easiest ways to realize parallel processing [3].

Thus today's parallel computers including virtual one have enough power to solve a large scale and complicated problem that was considered impossible a few years ago. With these progresses of hardware, software is also a versatile element for a parallel computer to realize a high performance, but it is always behind the hardware. While many researches are being done these years, more applications or techniques for a parallel computer are needed to bring out the ability of a parallel computer.

In this papaer, considering such trends of computing technology and requirements of solving large scale and complicated problems, a parallel FEM system which adopts a domain decomposition method (DDM) is developed and implemented on MPP and also on WSC. It is demonstrated that the developed system can solve a three-dimensional structural problem of over one million d.o.f. (degrees of freedoms) in a high parallel efficiency.

2 Domain Decomposition Method

For finite element analysis, a variety of parallel computing algorithms for a large scale problem have been studied by several researchers. Most of which take into account the node- or the element-wise, the column-wise, the domain-wise [4] concurrency, or their combination.

The domain-wise concurrency is found in the parallel substructure equation solvers [4–10] or in some domain decomposition methods [10–20]. It is well known that the parallel algorithm based on the domain-wise concurrency has, in general, a large granularity of parallel tasks.

To achieve a high performance which does not depend much on computer architecture, a large granularity of tasks and a well-balanced workload distribution have been key issues.

The Domain Decomposition Method, which the present study is based on, is originated in the well-known Schwarz method for solving elliptic problems. Although the original method is substantially of a sequential type, Glowinski et al. extended it to a parallel algorithm [11]. Their formulation, however, involves fully Neumann type calculations, which often generates floating domains that do not have enough prescribed displacements to eliminate the local rigid body modes.

In this paper, a new domain decomposition [18–20] formulation based on a displacement-based weighted residual method, which can avoid the fully Neumann type calculation is described.

2.1 Fundamental Equations using Lagrange Multipliers

The present DDM is summarized in the following. To explain its theory, let us consider an elastic problem concerning a domain Ω, as shown in Figure 1. Here, $\overline{\mathbf{T}}$ is the traction force applied on the boundary $\Gamma_{\mathbf{T}}$, $\overline{\mathbf{B}}$ the body force applied in the domain Ω, and $\overline{\mathbf{u}}$ the prescribed displacement on the boundary $\Gamma_{\mathbf{u}}$.

Fundamental equations of this elastic problem are summarized in an infinitesimal displacement mode as follows:

$$\varepsilon_{ij} = \frac{1}{2}(u_{i,j} + u_{j,i}) \quad \text{in } \Omega \tag{1}$$

$$\sigma_{ij} = C_{ijmn}\varepsilon_{mn} \quad \text{in } \Omega \tag{2}$$

$$\sigma_{ij,j} + \overline{B}_i = 0 \quad \text{in } \Omega \tag{3}$$

$$\sigma_{ij}\nu_j - \overline{T}_i = 0 \quad \text{on } \Gamma_{\mathbf{T}} \tag{4}$$

$$u_i = \overline{u}_i \quad \text{on } \Gamma_{\mathbf{u}} \tag{5}$$

where i,j take the value 1 to 3, u_i is a displacement vector, ε_{ij} a strain tensor, σ_{ij} stress tensor, C_{ijmn} a coefficient tensor of the Hooke's law and ν_j an outer normal vector on the boundary Γ, respectively. $()_{,j}$ denotes the first order derivative with respect to the coordinate x_j.

The above variational form is equivalent to the following minimization problem which finds the displacement function u which is a stationary point of the energy functional:

$$J(v) = \frac{1}{2}\int_\Omega \sigma_{ij}\varepsilon_{ij}d\Omega - \int_\Omega \overline{B}_i v_i d\Omega - \int_\Gamma \overline{T}_i v_i d\Gamma \tag{6}$$

As shown in Figure 2, after dividing domain Ω into N_d subdomains, $(\Omega^{(d)})_{1\leq d\leq N_d}$ with γ_{pq} being the interface between $\Omega^{(p)}$ and $\Omega^{(q)}$, solving the above problem is equivalent to finding the displacement functions $u^{(d)}$ which are stationary points of the energy functional:

$$J'(v^{(1)},\ldots,v^{(N_d)}) = J^{(1)}(v^{(1)}) + J^{(2)}(v^{(2)}) + \cdots + J^{(N_d)}(v^{(N_d)}) \tag{7}$$

with additional conditions on the interface boundary γ_{pq}:

$$u^{(p)} = u^{(q)} \quad \text{on } \gamma_{pq} \tag{8}$$

$$\sigma_{ij}^{(p)}\nu_j^{(p)} + \sigma_{ij}^{(q)}\nu_i^{(q)} = 0 \quad \text{on } \gamma_{pq} \tag{9}$$

where the superscripts $()^{(d)}$ designate variable defined in the subdomains $\Omega^{(d)}$.

Depending on the treatment of additional interface boundary conditions of equations (8) and (9), the following two approaches are available : In the first approach, equation (8) is satisfied exactly, while equation (9) is approximately satisfied, and vice versa in the second approach.

Although both the approaches are valid in principle, the first formulation involves fully Neumann type calculations, which often generates floating domains that do not have enough prescribed displacements to eliminate the local rigid body modes. The second approach is thus thought to be more appropriate.

With the use of a Lagrange multiplier method, solving the equation (7) with the subsidiary condition (9) is equivalent to finding the saddle-point of the Lagrangian functional:

$$\mathcal{L}(v^{(1)},\ldots,v^{(N_d)},\mu^{(1)},\ldots,\mu^{(N_i)}) = \sum_d^{N_d} J^{(d)}(v^{(d)}) + \sum_{p,q}^{N_d}\int_{\gamma_{pq}} \mu^T C(v^{(p)},v^{(q)})d\gamma \tag{10}$$

where

$$C(v^{(p)},v^{(q)}) = \sigma_{ij}^{(p)}\nu_j^{(p)} + \sigma_{ij}^{(q)}\nu_i^{(q)} \tag{11}$$

and N_i is total d.o.f. on interface γ_{pq}. The above problem can be equivalently converted to finding the displacement functions $u^{(d)}$ and the interface Lagrange multipliers $\lambda^{(i)}$ that satisfy:

$$\begin{aligned}\mathcal{L}(u^{(1)},\ldots,u^{(N_d)},\mu^{(1)},\ldots,\mu^{(N_i)}) &\leq \mathcal{L}(u^{(1)},\ldots,u^{(N_d)},\lambda^{(1)},\ldots,\lambda^{(N_i)})\\ &\leq \mathcal{L}(v^{(1)},\ldots,v^{(N_d)},\lambda^{(1)},\ldots,\lambda^{(N_i)})\end{aligned} \tag{12}$$

for any admissible $(v^{(d)})_{1\leq d\leq N_d}$ and $(\mu^{(i)})_{1\leq i\leq N_i}$.

We can solve this saddle-point problem (12) by a saddle-point solver such as Uzawa's algorithm or its CG variants. Let us describe the CG method to solve the equation (12) in next section.

2.2 Conjugate Gradient Algorithm for DDM

Defining the positive definite and symmetric operator $\mathcal{A}$:

$$\mathcal{A}\mu^{(i)} = C(u^{(p)}(\mu), u^{(q)}(\mu)) \tag{13}$$

where

$$u^{(p)}(\mu) = u^{(q)}(\mu) = \mu^{(i)} \text{ on } \gamma_{pq} \tag{14}$$

the CG algorithm for solving equation (12) is summarized as follows:

Step 0: Initialization

$$\mu^{(i)^0} \quad : \quad \text{arbitrarily given} \tag{15}$$

$$g^{(i)^0} = \mathcal{A}\mu^{(i)^0} \tag{16}$$

$$w^{(i)^0} = g^{(i)^0} \tag{17}$$

The $g^{(i)^0}$ of equation (16) is obtained from the traction forces on γ_{pq} which are calculated by solving equations (1)-(5) in each subdomains with the following constraint:

$$u^{(p)} = u^{(q)} = \mu^{(i)^0} \text{ on } \gamma_{pq} \tag{18}$$

Step 1: Steepest descent

$$\mu^{(i)^{n+1}} = \mu^{(i)^n} - \rho^n w^{(i)^n} \tag{19}$$

where

$$\rho^n = \frac{\sum_i^{N_i} g^{(i)^n} g^{(i)^n}}{\sum_i^{N_i} w^{(i)^n} \mathcal{A} w^{(i)^n}} \tag{20}$$

Step 2: Calculation of the new descent direction

$$g^{(i)^{n+1}} = g^{(i)^n} - \rho^n \mathcal{A} w^{(i)^n} \tag{21}$$

$$w^{(i)^{n+1}} = g^{(i)^{n+1}} + \kappa^n w^{(i)^n} \tag{22}$$

where

$$\kappa = \frac{\sum_i^{N_i} g^{(i)^{n+1}} g^{(i)^{n+1}}}{\sum_i^{N_i} g^{(i)^n} g^{(i)^n}} \tag{23}$$

The $\mathcal{A}w^{(i)^n}$ of equations (20) and (21) is obtained from the traction forces on γ_{pq} which are calculated by solving the following equations:

$$\sigma_{ij,j}{}^{(d)} = 0 \quad \text{in } \Omega^{(d)} \tag{24}$$

$$\sigma_{ij}{}^{(d)} \nu_j{}^{(d)} = 0 \quad \text{on } \Gamma_{\mathrm{T}}{}^{(d)} \tag{25}$$

$$u_i{}^{(d)} = 0 \quad \text{on } \Gamma_{\mathrm{u}}{}^{(d)} \tag{26}$$

$$u_i{}^{(d)} = w^n \quad \text{on } \gamma_{pq} \tag{27}$$

Step 3: Judgment of convergence

If $\mu^{(i)^n}$ has not converged yet, return to Step 1 by setting n to be $n+1$. Here the convergence criterion is defined as:

$$\frac{\max_i |g^{(i)^n}|}{\max_i |g^{(i)^0}|} < Err \tag{28}$$

in which the maximum component of force imbalance along the interface boundary, i.e. residual value, is monitored.

The flow chart of the present DDM algorithm is illustrated in Figure 3. It should be noted here that the finite element analysis of each subdomain can be performed without any data communication among subdomains. Namely, the finite element analyses of subdomains can be performed in parallel, once the displacement values on the inter-subdomain boundaries are given. Since the workload for each finite element calculation is much larger than those of other tasks including data communication and modification of boundary values, the so-called overhead due to parallel calculation is estimated to be very small. In addition, owing to the decomposition of a large scale finite element system into a number of smaller sub-systems, only small computation storage is needed for each finite element calculation.

2.3 Preconditioning for CG

In matrix formulation, problem (6) are given by:

$$Ku = f \tag{29}$$

where K, u, and f are respectively the stiffness matrix, the displacement vector and the force vector associated with the finite element discretization of Ω.

For each subdomain problems of equations (24)-(27), the matrix formulations are written as:

$$K^{(d)}u^{(d)} = f^{(d)}, \ d = 1, 2, \ldots, N_d \tag{30}$$

Let us use the i and b subscripts to designate internal and interface boundary d.o.f. and if the internal d.o.f. first and the interface boundary d.o.f. are numbered last, equation (30) can be written as:

$$\begin{bmatrix} K_{ii}^{(d)} & K_{ib}^{(d)} \\ K_{ib}^{(d)T} & K_{bb}^{(d)} \end{bmatrix} \begin{bmatrix} u_i^{(d)} \\ u_b^{(d)} \end{bmatrix} = \begin{bmatrix} f_i^{(d)} \\ f_b^{(d)} \end{bmatrix} \tag{31}$$

that is:

$$K_{ii}^{(d)}u_i^{(d)} + K_{ib}^{(d)}u_b^{(d)} = f_i^{(d)} \tag{32}$$

$$K_{ib}^{(d)^T}u_i^{(d)} + K_{bb}^{(d)}u_b^{(d)} = f_b^{(d)} \tag{33}$$

From equation (32):

$$u_i^{(d)} = K_{ii}^{(d)^{-1}}(f_i^{(d)} - K_{ib}^{(d)}u_b^{(d)}) \tag{34}$$

From equations (33) and (34):

$$(K_{bb}^{(d)} - K_{ib}^{(d)^T}K_{ii}^{(d)^{-1}}K_{ib}^{(d)})u_b^{(d)} = f_b^{(d)} - K_{ib}^{(d)^T}K_{ii}^{(d)^{-1}}f_i^{(d)} \tag{35}$$

Considering the condition (24), equation (35) is written as:

$$S^{(d)}u_b^{(d)} = f_b^{(d)} \tag{36}$$

where

$$S^{(d)} = K_{bb}^{(d)} - K_{ib}^{(d)^T}K_{ii}^{(d)^{-1}}K_{ib}^{(d)} \tag{37}$$

$S^{(d)}$ is known as the local Schur complement [21].

Now the operator $\mathcal{A}$ of equations (20) and (21) is written as:

$$\mathcal{A} = \sum_{d}^{N_d} B^{(d)}S^{(d)}B^{(d)^T} \tag{38}$$

where $B^{(d)}$ is a boolean symbolic matrix which localizes a internal quantity to the interface boundary.

For the above operator, a good preconditioner P [22] can be constructed by assembling the primal subdomain operators as follows:

$$P^{-1} = \sum_{d}^{Nd} B^{(d)} \begin{bmatrix} 0 & 0 \\ 0 & K_{bb}^{(d)} - K_{ib}^{(d)^T} K_{ii}^{(d)^{-1}} K_{ib}^{(d)} \end{bmatrix} B^{(d)^T} \tag{39}$$

and it's lumped preconditioner P' is advocated in [23], that is:

$$P'^{-1} = \sum_{d}^{Nd} B^{(d)} \begin{bmatrix} 0 & 0 \\ 0 & K_{bb}^{(d)} \end{bmatrix} B^{(d)^T} \tag{40}$$

This preconditioner does not require any additional storage and involves only matrix-vector products of sizes equal to the interface.

3 Implementation on MPP

As mentioned in the previous section, a large granularity of tasks can be obtained through the DDM. To achieve high performance of parallel algorithms, a technique to balance workload well among processors is demanded.

With a SIMD type of MPP, in order to do so, we must divide a domain of concern and provide and a uniform distribution with data to all processors in a regular fashion, although for a complicated problem, it is not easy.

To keep the load balance for an irregular decomposition problem such as a complicated model, dynamic data allocation, which can be performed by MIMD type of MPP, is desirable. With SIMD type of MPP, only data is distributed among processors, while with MIMD type of MPP, instructions can be also distributed, that is, it can perform more advanced or complicated parallel technique such as the dynamic data allocation.

This fact thus motivated us to use the MIMD type of MPP with dynamic data allocation technique, and in addition for a large size problem, to avoid a limitation from the memory size of each processors, the hierarchical data management technique is developed and implemented on MPP.

3.1 Data Allocation

In each sub structuring or domain decomposition method, a physical problem, i.e. a domain to be analyzed, is fictitiously divided into a number of subdomains. There are two approaches in allocating parallel tasks, i.e. calculations of subdomains, on multiple processors. The first approach is the so-called "Static Workload Balancing" and the second is the "Dynamic Workload Balancing".

In static workload balancing, task allocation is performed a priori considering workload balance among processors.

For simplicity, let us consider a problem which is divided into nine subdomains and allocated to nine processors as shown in Figure 4. Considering how to allocate the divided data to each processor, a simple idea allows one subdomain data to be allocated to one processor when the numbers of subdomains and processors coincide. Therefore, in static workload balancing, it is important to divide domain into the same number of subdomains with processors.

With this method, when all subdomain's data are equal in size and the abilities of all the processors are equivalent, all processors can start and end the analysis of each subdomain at the same time and then change needed data among processors and start the next iteration step. In such a case, as shown in Figure 5 wherein the vertical and horizontal axes denotes the number of processors and time, respectively, well workload balancing can be obtained easily.

Such a static workload balancing process is easy to implement when the domain can be divided regularly into the same number of subdomains with processors on a parallel computer which have an identical performance like MPP. However, it is often troublesome, and does not always work well when the geometry of the domain is complicated or each processor has a different performance as can be seen in a clustered of different workstations.

When each size of subdomains are irregular (Figure 6) or processor's abilities are different, workload balance can not be obtained similarly to the above case and it is getting lack of balance as shown in Figure 7.

To handle this problem, another technique is needed to allocate the task to processors well, which is so-called the dynamic workload balancing technique, and is described as follows.

In dynamic workload balancing, task allocation among processors is performed automatically and dynamically during calculation.

For an irregular decomposition problem, it is clear that allocating one subdomain to one processor can not obtain well workload balance as shown in Figure 7.

By combining small and large subdomains for each processor, the total workload balance can be attained well as shown in Figure 8,

When the calculation time of each subdomain can be estimated prior to the calculation, task allocation can be performed beforehand statically, but usually it is not easy to estimate accurately and sometime it is changed during calculation. Therefore, to perform such an allocation, the dynamic allocation is needed.

In dynamic allocation, data of another subdomain is provided as soon as the analysis of one subdomain is finished. An efficient workload is thus obtained if the whole domain is decomposed into a large number of subdomains in comparison to the number of processors In Figure 8, twenty domains are analyzed with nine processors where each processor deals with two or three subdomains.

To take advantage of such dynamic workload balancing, the parallel finite element algorithm described in chapter 2 is implemented on the MIMD type of MPP.

3.2 Roles among Processors

To implement the DDM which incorporates the dynamic workload balancing on the MIMD type of MPP, we have to provide some roles to each processors. To allocate the data of subdomains to processors, some processors have to manage data. That is, we need manager processors and analyzer processors. With the MIMD of MPP, such tasks can be coordinated among processors.

Now, there are two types of management system, each consisting of either a manager and analyzers set or one chief manager 'Father-Child Model', some managers 'Grand-Father-Child Model' and analyzers set. The latter is required for a large scale model which can not be managed by only one manager processor. These two types are described as follows.

In 'Father-Child Model' system, one processor is set as a manager which is called 'Father' and the others each as a analyzer which is called 'Child'. Figure 9 shows the schematic data flow among processors of the present system. The role of the 'Father' is to manage data and that of the 'Child' is to execute finite element analysis of each subdomain. The data flow among these two kinds of processors are summarized as follows.

The 'Father' reads mesh data which is previously divided into some subdomains and prepares initial values on the interface boundaries. After reading data from a disk 'Father' provides subdomain's data to any idling 'Child'.

Now, any 'Child' can receive any subdomain's data to reduce the idling time. As mentioned in previous section , if subdomain's data which are not of regular size are allocated to 'Child's statically, some 'Child's have to wait until other 'Child' which is allocated to a larger subdomain complete the analysis of the subdomain. To handle this problem, in this system, the 'Child' allocated to a smaller subdomain can get the next unsolved subdomain's data as soon as the previous analysis is over.

The 'Child' which receives subdomain's data from a 'Father' executes the finite element analysis of the subdomain and, after the analysis, sends its result to the 'Father'. After receiving all the results of the subdomains, the 'Father' adjusts the values on the interface boundaries so as to hold the balance among subdomains. These operations are iterated until the convergence is achieved.

Figures 10 and 11 show the flow chart of 'Father' and 'Child' processors, respectively. Owing to the present data flow mechanism, the whole workload of analysis can be well-balanced among all the processors dynamically as well as automatically.

In the previous system, all the data is first read by one 'Father' processor. Since in the case of large scale problem, the data exceeds one processor's memory, the system obviously has a limitation in analysis in terms of its memory size. To handle this problem, in 'Grand-Father-Child Model' system, one processor is set as a chief manager which is termed 'Grand', some processors as a manager termed 'Father' and the others as a analyzer termed 'Child'.

Figure 12 shows the schematic data flow among processors of the present system. The role of the 'Grand' is to manage values on the interface boundaries and status of 'Father' processors, that of the 'Father' is to store and manage data while 'Child' executes the finite element analysis of each subdomain. The data flow among these three kinds of processors are summarized as follows.

The 'Grand' prepares initial values on the interface boundaries. Each 'Father' reads the whole mesh data previously divided into the same number of parts as 'Father's and each part data is divided into some subdomains. After reading the data from a disk 'Father' provides subdomain data to any idling 'Child'. The 'Child' which receives subdomain's data from a 'Father' executes the finite element analysis of the subdomain and after the analysis sends its result to the 'Father'.

Now, every 'Child' can receive any subdomain's data in the same way as the previous method, and communicate with any 'Father' to reduce the idling time. That is, if one of 'Father's has no data for analysis, 'Child' which previously received data from the 'Father' can ask to another 'Father' for more data. As long as the 'Father' has the data of the subdomain for analysis, any 'Child' keeps working without idling. After receiving all the results of subdomains, the 'Father' sends results of interface boundaries to the 'Grand'. The 'Grand' then adjusts the values on the interface boundaries so as to hold the balance among subdomains. These operations are iterated until the convergence is achieved.

Owing to the present data flow mechanism, the proposed system can avoid a limitation of problem's size which depended upon the memory size of the father processor, and the whole workload of analysis can be well-balanced among all the processors dynamically as well as automatically.

3.3 Hierarchical DDM

In this parallel DDM, as shown in Figure 3, calculation time is in proportion to the number of CG iterations. In CG method, its iteration number much depends on d.o.f. of problem, i.e. in this case d.o.f. on interface boundary. As the number of subdomains escalates with the scale of the problem, the calculation time increases simply as well as d.o.f. on interface boundary.

In CG method, there exists a limit of d.o.f., over which the iteration number increases rapidly. The limit depends on the complexity of the problem; i.e. for a large scale and complex problem, calculation time could be enormous. To reduce the CG iteration number, it is needed to divide problem into a small number of subdomains to reduce d.o.f. on the interface boundary. Since the size of each subdomain depends on the number of subdomains, decreasing the subdomain's number causes increase of the subdomain's size. Each subdomain's size is, however, restricted in terms of the memory size of a processor which analyze the subdomain. To handle the above problem for a large scale analysis, a hierarchical DDM is developed in this paper, as described in the following.

In this new method, a whole analysis domain is first divided into a number of large subdomains in a rough manner. To do this, the total d.o.f. on the interface boundary can be set small, but each subdomain's size is over the domain that can be analyzed with one processor. To analyze each subdomain, each subdomain is divided hierarchically into a number of sub-subdomains and apply the DDM to each subdomain. It means, in each subdomain, FEM analysis is performed by the DDM with iterative calculation. After achieving convergence in all subdomains, the values on the interface boundaries of all the subdomains are adjusted until the convergence in the whole domain is achieved.

Notwithstanding the double loop of CG iteration, the hierarchical division can decrease d.o.f. on interface boundary under keeping all the subdomains small in size. The flow of the hierarchical DDM algorithm is illustrated in Figure 13.

Table 1: Mesh sizes for pressure vessel analysis model

Model	Elements	Nodes	Total d.o.f.	Domains	Interface d.o.f.
1	37,537	66,796	200,388	586	66,699
2	220,245	348,369	1,045,107	6,076	543,663

4 Analysis of Large Scale Elastic Problem

For solving a large scale model under the limitation of memory size, using very rough mesh which cannot provide satisfied results or analyzing small part of the domain separately are the ways to escape from its size problem. As supercomputers have enormous size of memory, this restriction is getting removed and such a large scale model is becoming realistic for analysis though it was considered impossible a few years ago. However, requirements for numerical simulation increases much faster than the speedup of such supercomputers. The MPPs have the ability for large scale analysis more greatly than single processor computers, but such hardware needs appropriate software.

With the present parallel FEM system, not only it can achieve high parallel performance, but also it does not much depend on hardware. That is, it has ability to apply to any kind of parallel computer with a high performance. As an application of the present system to a large scale and complicated model, a whole pressure vessel with nozzles model is analyzed in this chapter. The problem is solved with some different of parallel computer including workstation clusters to estimate the robustness of the system.

4.1 Pressure Vessel Model

The geometry of the model is defined and we assume that the inner surface of the model is under pressure. The model was expressed by 10-noded tetrahedron elements with the density of mesh around nozzles being set higher than the other part. For this model, two sizes of mesh are generated, models 1 and 2, sizes of which are listed in Table 1.

Each model is divided into subdomains with parameters listed in Table 1. This division is determined from the capacity of the memory of 'Child' processor. To analyze the same problem with different computers, the memory takes the value of the smallest processor's memory, which amounts to 4 Mbytes memory. Since it is small, the number of divisions has become large which causes a large number of interface d.o.f.

In dividing into parts, the division number of parts depends on the memory size of 'Father' processor and since the number does not have effect on the number of CG iterations and since sometimes it is not needed to divide into parts, i.e. in case of simpler "Father-Child Model" is available, it is set adaptive to a computer to be used.

Figure 14 shows an example for a mesh which is divided into 4 parts.

4.2 Comparison between Different Computers

To compare performance between different computers, the model 1 was analyzed using computers listed in Table 2. The nCUBE2, T3D and CM5 are commercial MPP, whereas WSC and PCs are clustered computers substantially forming a parallel computer. The WSC is a workstation cluster which consists of nine workstations having 15 processors wherein six workstations each fine have two processors inside. The PCs are a personal computer cluster which consists of five personal computers. These computers listed in Table 3 were connected through a network. The WSC was connected through 10Base5 Ethernet, peak performance of which is 10 Mbps (Mega bit per second), and the PCs are connected through 10BaseT Fast-Ethernet whose performance is 10 times higher than the WSC's.

In Tables 2 and 3, as a yardstick of performance, it is referred to the LINPACK Benchmark [24] and estimated communication time between processors. On nCUBE2, their original communica-

Table 2: Performance using some different computers

Model	Num. of Proc.	Memory [Mbytes/Proc.]	LINPACK [Gflop/s]	Comm. Time [Mbytes/sec]
nCUBE2	1,024	4	1.9	2
T3D	256	64	25.3	24
CM5	32	16	1.9	5
WSC	15	64-256	-	0.5
PCs	5	64	-	2.5

Table 3: Workstation list of WSC

Model	Num. of Proc.	LINPACK [Mflop/s]
HP 9000/735(99MHz)	2	41
HP 9000/715(33MHz)	1	12
Sun SS20/612(60MHz)	8	12
Sun SS10/512(50MHz)	4	10
Pentium(133MHz)	5	16

tion library was used and on the others PVM library [25] was utilized to communicate between processors.

Calculation time of 2 CG iterations on these computers and parallel performances are shown in Table 4. In the case of nCUBE2, one processor was assigned as 'Grand', seven processors as 'Father' and the other processors, 24 or 248 as 'Child' considering small local memory size. For the other MPP's cases, one processor was assigned as 'Father' and the other processors as 'Child'. On the WSC and PCs, one processor assigned as 'Father' as well as 'Child' using the time sharing system which permitted to be used by several tasks at the same time and the other processors as 'Child'.

As a parallel performance, the rate of CPU usage was used. The rate of CPU usage R_n with n processors is defined as follows:

$$R_n = \frac{1}{n}\sum_{i}^{n}\frac{T_{work}^{(i)}}{T_{work}^{(i)} + T_{idle}^{(i)}} \tag{41}$$

where $T_{work}^{(i)}$ and $T_{idle}^{(i)}$ are the total time for working and idling of each processor during the whole computation, respectively. As shown in Table 4, high parallel performances over 90 % are achieved for all the cases except two cases, i.e. cases of nCUBE2 with 32 processors and WSC. In the former case, the reason is that number of 'Father's is too large compared with number of 'Child's. Seeing CPU usage among only 'Child' processors, it achieved over 99%. In the latter case, the low performance result from the slow communication time using Ethernet.

4.3 Sorting Technique

To demonstrate the dynamic allocation system described in section 3.1, the model 2 was solved again with the nCUBE2 consists of 128 processors. In this example, one processor was assigned as 'Grand', seven as 'Father' and 120 as 'Child'.

Figure 15 shows the time chart of working states of 120 'Child' processors during two different iterative steps of CG iterations. In this figure, the length of each horizontal bar indicates the analysis period of a subdomain and the blanks indicate the idling time of 'Child' processor waiting subdomain's data to analyze from 'Father' processor. The seven color means in which 'Father' subdomain data is stored. That is, the subdomain of magenta color is managed and provided by

Table 4: Calculation times of 2 CG iterations and parallel performances

Model	Num. of Proc.	Num. of (G+)F+C	CPU time[sec]	CPU Usage[%]
nCUBE2	32	1+7+24	773.0	74.5
	256	1+7+248	80.2	90.4
T3D	32	1+31	46.9	95.0
CM5	32	1+31	169.5	95.4
W.S.C.	15	1+15	85.4	85.3
PCs	5	1+5	197.3	96.2

magenta color 'Father'. For example, No.1 'Child' is provided from magenta 'Father' and No.18 'Child' is from red 'Father', first.

However, as described in section 3.2, the relation is just a default set, i.e. any 'Child' can get data from any 'Father'. For example, No.1 'Child' is provided from first nine subdomains of magenta 'Father', but when it finishes the ninth subdomain, the magenta 'Father' can not provide next data because he has no more unsolved data, so No.1 'Child' requires to another 'Father' and then can get the next data from the blue 'Father'.

In this figure, there is still a wide space of black which causes efficiency to be low. The reason is that the subdomain which is finished to be analyzed last has a larger size. Sorting subdomains data from large to small, the larger subdomains can be analyzed in the first period of one iteration step. Figure 16 shows the new time chart after sorting subdomain's data. As shown in the figure, it clearly demonstrates that the work load balance among processors is fulfilled automatically and dynamically, thanks to the sorting technique.

In the model 1 case, one size of each subdomain is small enough compared with the total time of each iteration step, even without the sorting technique, it can perform with high efficiency.

4.4 Analysis of Pressure Vessel Model

For model 1, the nCUBE2 consists of 1,024 processors was used to solve. As shown in Table 2, the memory size of nCUBE2 is 4 Mbytes and because of it, the model was divided into 25 parts to be read by such processors. Therefore, one was assigned as 'Grand', 25 as 'Father' and the rest 998 as 'Child'.

To estimate the convergence, the variations of the residual value, defined in equation (28), against the number of the CG iterations of the two models 1 and 2 are shown in Figure 17 and 18, respectively. In each model, two cases are plotted, i.e. using normal CG algorithm and preconditioning CG algorithm which is described in section 2.3. As shown in these figures, preconditioning technic is useful and necessary.

To check the convergence of a physical value for this model, a small scale of this model, which has 71,022 d.o.f., was analyzed. Figures 19 and 20 show the variations of the residual value and the displacement of one position which is on the inner surface of the vessel with respect to the number of the CG iterations, respectively. As shown in these figures, despite the fact that the residual values vibrate locally, the displacement value converges with the number of iterations.

For model 1, over one million d.o.f. problem has been solved with 1,024 processors, 6,698 iterations, 182 hours and 96.5% parallel performance.

5 Conclusions

The parallel finite element system based on the DDM, which works on a massively parallel computer were successfully developed in the present study.

This system can be applied to several types of parallel computers including clustered workstations or personal computers with high parallel performance.

This system combined with preconditioning for CG was applied to a pressure vessel with nozzles model with nonuniform mesh and irregular domain decomposition of over one million d.o.f. and a parallel performance of over 90 with 1,024 CPUs was obtained.

References

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

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

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

[4] C.Farhat and L.Crivelli, A general approach to nonlinear FE computations on shared-memory multiprocessors, *Computer Methods in Applied Mechanics and Engineering* **72** (1989) 153-171.

[5] C.Farhat and E.Wilson, A new finite element concurrent computer program architecture, *International Journal for Numerical Methods in Engineering* **24** (1987) 1771-1792.

[6] O.O.Storaasli and P.Bergan, A nonlinear substructuring method for concurrent processing computers, *AIAA Journal* **25** (1987) 871-876.

[7] C.Farhat, E.Wilson and G.Powell, Solution of finite element systems on concurrent processing computers, *Engineering with Computers* **2** (1987) 157-165.

[8] J.H.Hajjar and J.F.Abel, Parallel processing for transient nonlinear structural dynamics of three-dimensional framed structures using domain decomposition, *Computers and Structures* **30** (1988) 1237-1254.

[9] J.Hajjar and J.Abel, On the accuracy of some domain-by-domain algorithms for parallel processing of transient structural dynamics, *International Journal for Numerical Methods in Engineering* **28** (1989) 1855-1874.

[10] I.S.Doltsinis and S.Noelting, Studies on parallel processing for coupled field problems, *Computer Methods in Applied Mechanics and Engineering* **89** (1991) 497-521.

[11] R.Glowinski, O.V.Dinh and J.Periaux, Domain decomposition methods for nonlinear problems in fluid dynamics, *Computer Methods in Applied Mechanics and Engineering* **40** (1983) 27-109.

[12] C.T.Sun and K.M.Mao, A global-local finite element method suitable for parallel computations, *Computers and Structures* **29** (1988) 309-315.

[13] A.K.Noor and J.M.Peters, A partitioning strategy for efficient nonlinear finite element dynamic analysis on multiprocessor computers, *Computers and Structures* **31** (1989) 795-810.

[14] R.Glowinski, G.H.Golub, G.A.Meurant and J.Periaux (eds), First international symposium on domain decomposition methods for partial differential equations, *SIAM. Philadelphia, PA* (1988).

[15] G.Yagawa, Parallel techniques for computational mechanics, *Theoretical and Applied Mechanics* **39** (1990) 3-9.

[16] J.C.Luo and M.B.Friedman, Implicit decomposition as a tool for solving large-scale structural systems in a parallel environment, *Computers and Structures* **35** (1990) 215-220.

[17] Y.Zhang and R.S.Harichandran, Implicit subdomain integration for dynamic analysis of large-scale structural systems, *Computer Methods in Applied Mechanics and Engineering* **81** (1990) 57-70.

[18] G.Yagawa, N.Soneda and S.Yoshimura, A large scale finite element analysis using domain decomposition method on parallel computer, *Computers and Structures* **38** (1991) 615-625.

[19] G.Yagawa, A.Yoshioka, N.Soneda and S.Yoshimura, A parallel finite element method with a supercomputer network, *Computers and Structures* **47** (1993) 407-418.

[20] G.Yagawa and R.Shioya, Parallel finite elements on a massively parallel computer with domain decomposition, *Computing systems in Engineering* **4(4-6)** (1994) 495-503.

[21] P.E.Bjorstad and O.B.Widlund, Iterative methods for the solution of elliptic problems on regions partitioned into substructures, *SIAM J. Numer. Anal.* **23(6)** (1986) 1097-1120.

[22] J.Mandel, Balancing domain decomposition, *Comm. Appl. Num. Meth.* 9 (1993) 233-241.

[23] C.Farhat and F.X.Roux, A method of finite element tearing and interconnecting and its parallel solution algorithm, *International Journal for Numerical Methods in Engineering* **32** (1991) 1205-1227.

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

[25] A.Beguelin, J.Dongarra, A.Geist, R.Mancheck and V.sunderam, A user's guide to PVM : Parallel virtual machine, *Technical Report, Mathematical Sciences Section, Oak Ridge National Laboratory* ORNL/TM-11826 (1991).

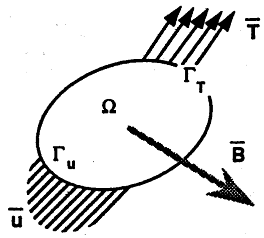

Figure 1: Analysis domain

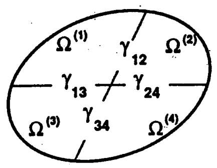

Figure 2: Analysis domain split into subdomains

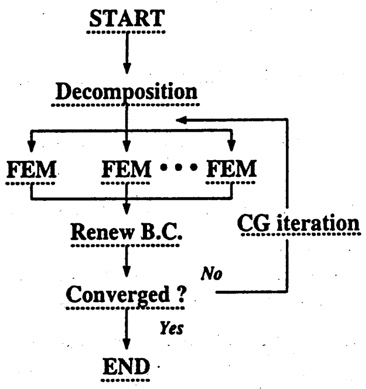

Figure 3: Flow chart of domain decomposition method

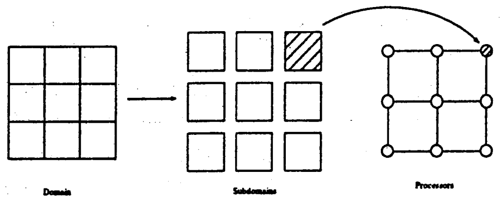

Figure 4: Static data allocation of regular subdomains

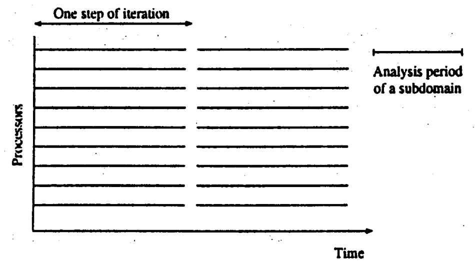

Figure 5: Regular workload balancing among processors

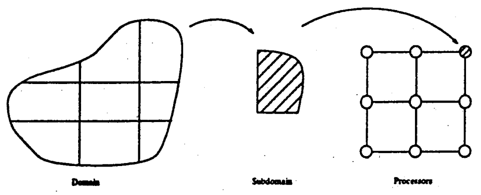

Figure 6: Static data allocation of irregular subdomains

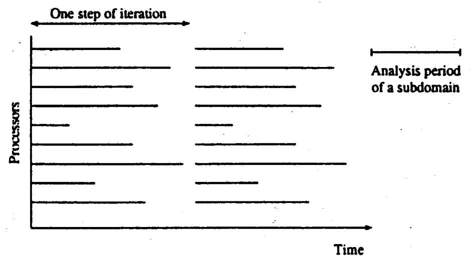

Figure 7: Irregular workload balancing among processors

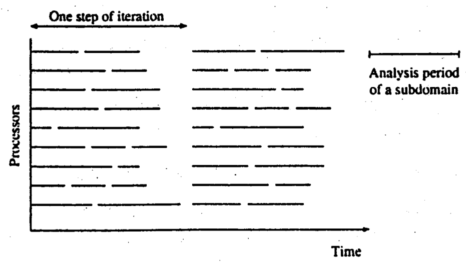

Figure 8: Dynamic workload balancing among processors

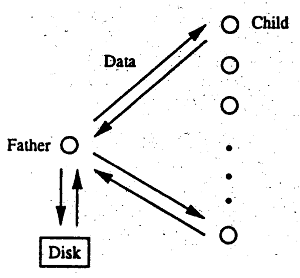

Figure 9: Schematic data flow among Father-Child processors

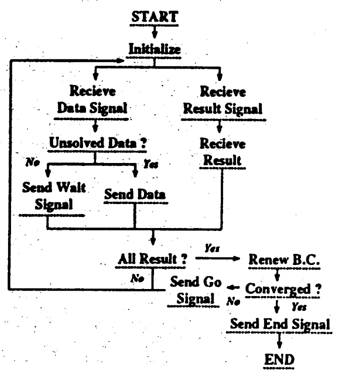

Figure 10: Flow chart of Father processor

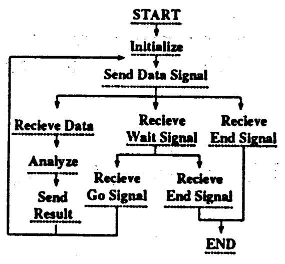

Figure 11: Flow chart of Child processor

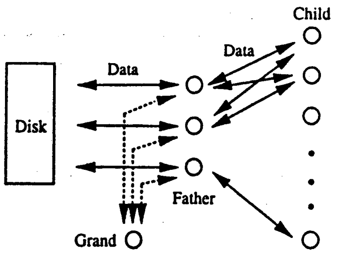

Figure 12: Schematic data flow among Grand-Father-Child processors

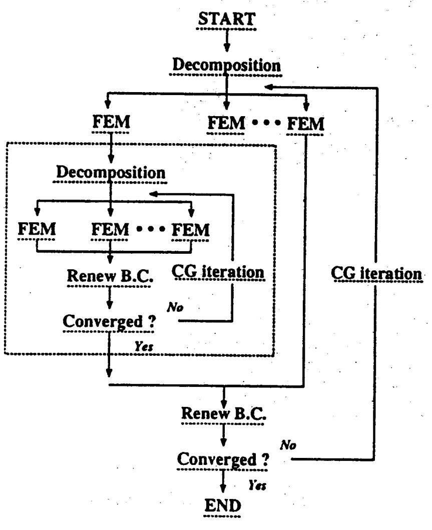

Figure 13: Flow chart of hierarchical DDM

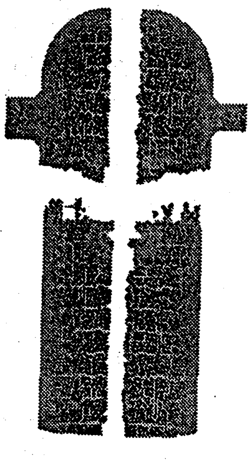

Figure 14: Pressure vessel model divided into parts

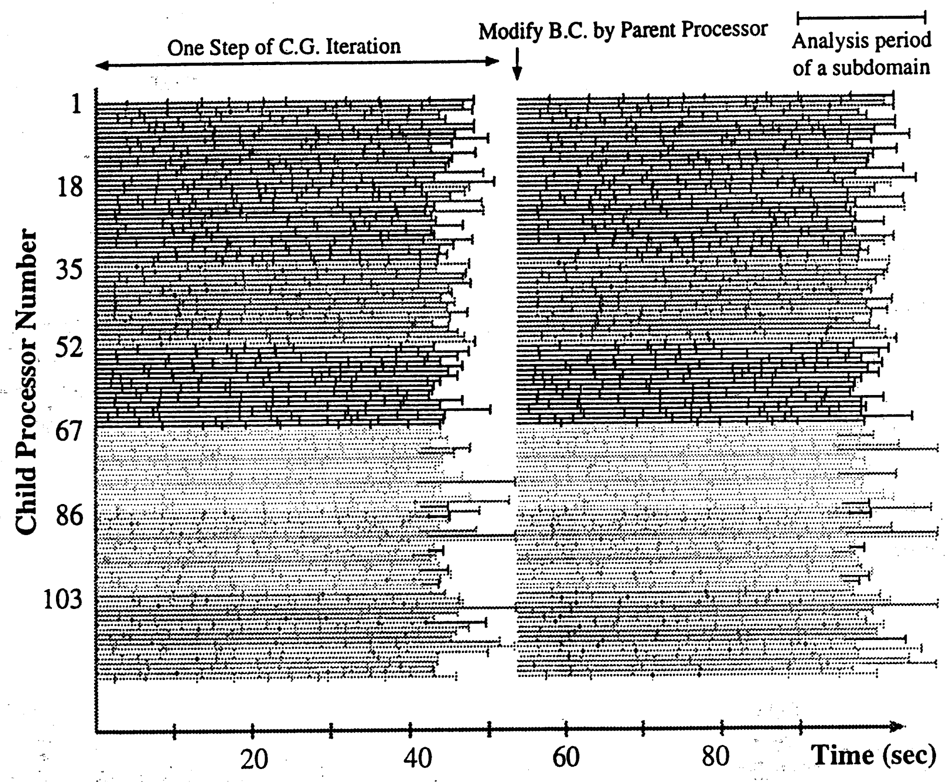

Figure 15: Time chart of working states for model 1 without sorting technique

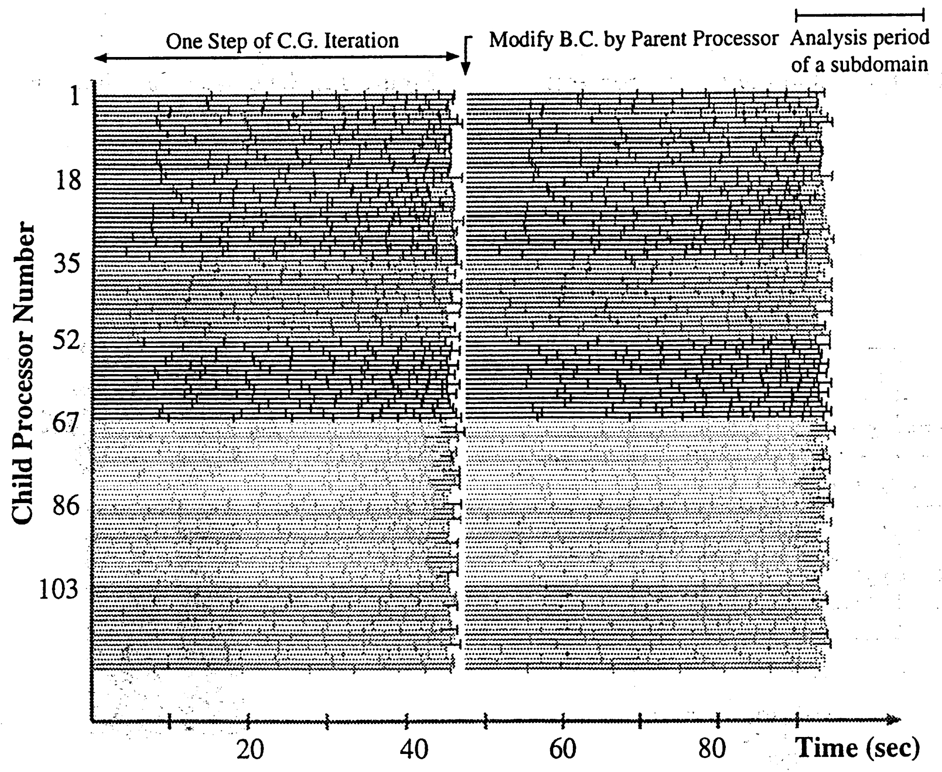

Figure 16: Time chart of working states for model 1 with sorting technique

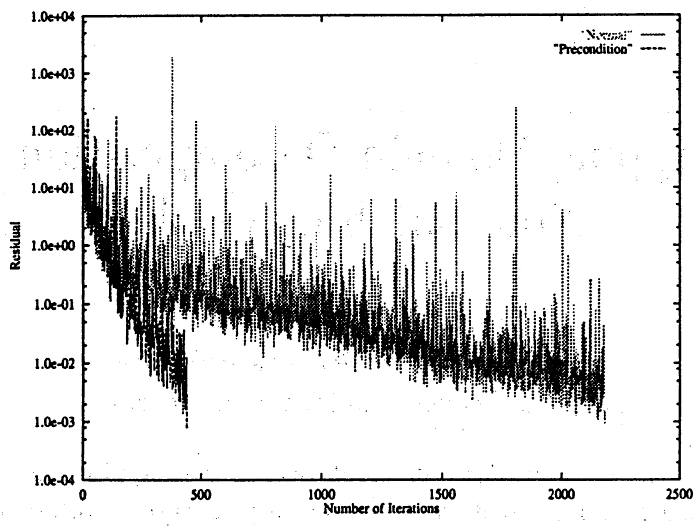

Figure 17: Residual vs number of iterations for model 1 with preconditioning

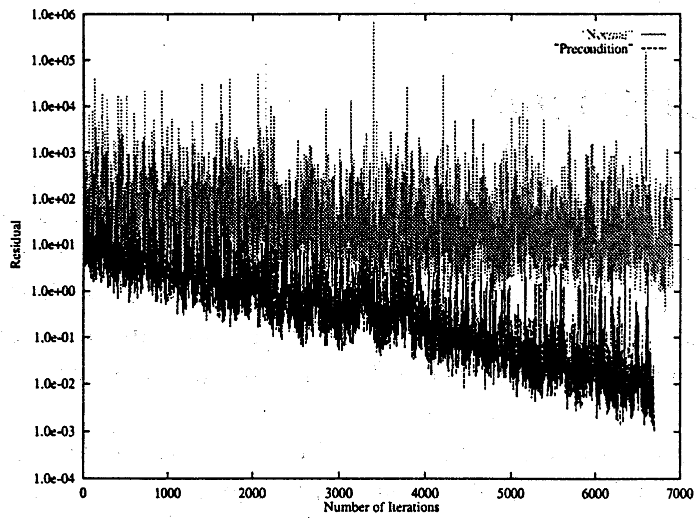

Figure 18: Residual vs number of iterations for model 2 with preconditioning

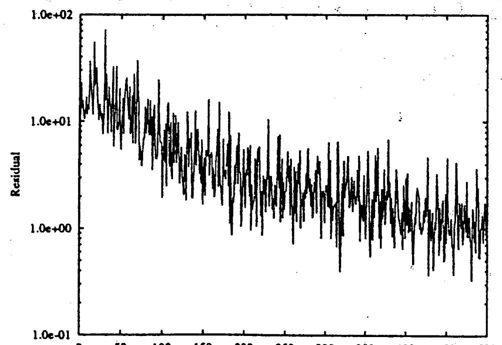

Figure 19: Residual vs number of iterations

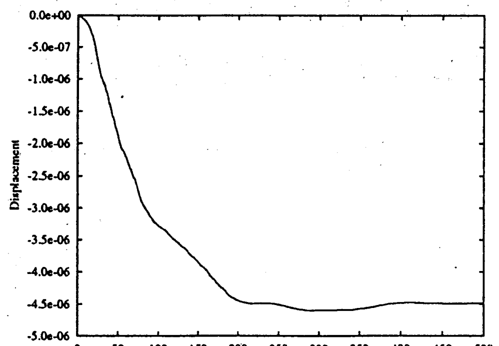

Figure 20: Displacement vs number of iterations