Finite Element Analysis with Heterogeneous Parallel Computer Environment over ATM Network

Hideo Fukumori, Yoichi Kono, Ken Nishimatsu, and Yoichi Muraoka
E-mail: fukumori@muraoka.info.waseda.ac.jp
School of Science and Engineering
Waseda University
3-4-1 Okubo Shinjyuku-ku, Tokyo, Japan
Phone: +81-3-3209-5198
Fax: +81-3-3209-5198

Abstract

In this paper, we present an implementation of FEM solver on heterogeneous parallel environment that consists of two different parallel computers (the Fujitsu AP1000 and NEC Cenju-3). We used OLU (On Line University) Network, which is one of the wide area ATM network connecting over 20 universities and research facilities all over Japan.

We used substructure method applied in multiple levels for the calculation algorithm. This algorithm has small data dependency in the calculation phase and the number of data transfer between the parallel computers is limited, thus the overhead for the synchronization can be smaller than iterative method.

This paper also refers to the parallel triangular mesh generator based on the Delaunay Triangulation currently being implemented on the Fujitsu AP1000 parallel computer.

1 OLU Network

OLU(On Line University) Network is a wide area network project using ATM and optic fiber technology. There are 23 sites in Japan and they are connected in ring topology. The theoretical bandwidth of this network is 156Mbps and actual bandwidth is 67Mbps.

Our implementation used the link between Waseda University(Shinjyuku, Tokyo) and NEC C&C Laboratory (Kawasaki, Kanagawa) (fig.1). There are five other sites in between, and the total physical network length is 370 kilometers.

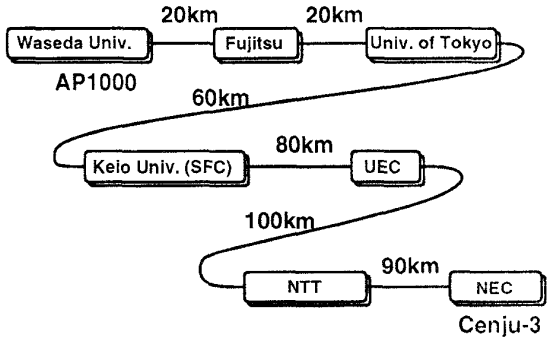

Figure 1: OLU Network Connection Between Waseda and NEC

2 Calculation Algorithm

Substructure method [1][2] applied in multiple levels[3][4] was used as the algorithm for the calculation process. Basically, this algorithm is the direct method used with domain decomposition. The calculation and assembly of the coefficient matrices is done in multiple levels and this results in producing small, dense matrices instead of one large sparse coefficient matrix. The quality of the dense matrix makes it easier to reduce the number of fill-ins and achieve efficient data exchange between the parallel computers over network.

Multi-level substructuring is done in the following steps:

1. Decompose the domain of analysis into the first level substructures that consists of 9 nodes. One

1087-4089/96 $5.00 © 1996 IEEE

node is placed inside of the substructure and the rest is on the outside edge (fig.2).

2. In the substructure level k,

   (a) Index the nodes inside the substructure, followed by the nodes on the boundary. This indexing results in the equation:

$$\begin{bmatrix} A_{ii}^{(k)} & A_{ib}^{(k)} \\ A_{bi}^{(k)} & A_{bb}^{(k)} \end{bmatrix} \begin{bmatrix} x_i^{(k)} \\ x_b^{(k)} \end{bmatrix} = \begin{bmatrix} b_i^{(k)} \\ b_b^{(k)} \end{bmatrix} \quad (1)$$

   (b) The upper and lower part represents the equations for the node inside the substructure and the nodes on the boundary of the substructure, respectively. Inserting the upper part of this equation into the lower part yields:

$$A_{bb}^{*(k)} x_b^{(k)} = b_b^{*(k)} \quad (2)$$

   (c) Substructures make pairs and add up $A_{bb}^{*(k)}$ and $b_b^{*(k)}$ between them to form $A_{bb}^{*(k+1)}$ and $b_b^{*(k+1)}$ (fig.3).

3. Repeat above steps until there is only one substructure left. Denote this highest substructure level as k_{max}.

4. Solve

$$A^{(k_{max})} x^{(k_{max})} = b^{(k_{max})} \quad (3)$$

   and obtain $x_i^{(k_{max})}$.

5. Substructures in the level $k-1$ obtain $x_i^{(k-1)}$ from the equation:

$$x_i^{(k-1)} = (A_{ii}^{(k-1)})^{-1}(b_i^{(k-1)} - A_{ib}^{(k-1)} x_b^{(k)}) \quad (4)$$

6. Repeat above processes until they reach the first substructure level, where $k = 1$.

The overall calculation process in multi-level substructuring can be shown in binary tree (fig.4). Data exchange between levels occurs in:

- the assembly of coefficient matrix (step 2.(c))
- the calculation for the final solution (step 5.)

If the size of the problem is n (nodes) and the perfect binary tree could be constructed, the order of the multiplication will be $O(n^{\frac{3}{2}})$ for the two-dimensional problems.

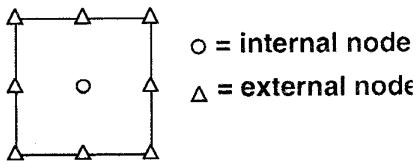

Figure 2: First Level Substructure

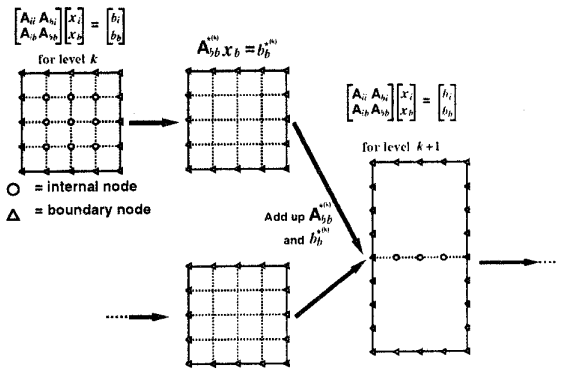

Figure 3: Calculation Process of the Multi-level Substructuring

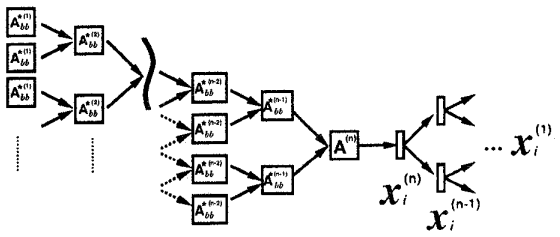

Figure 4: Assembly of A_{bb} and Redistribution of x_i

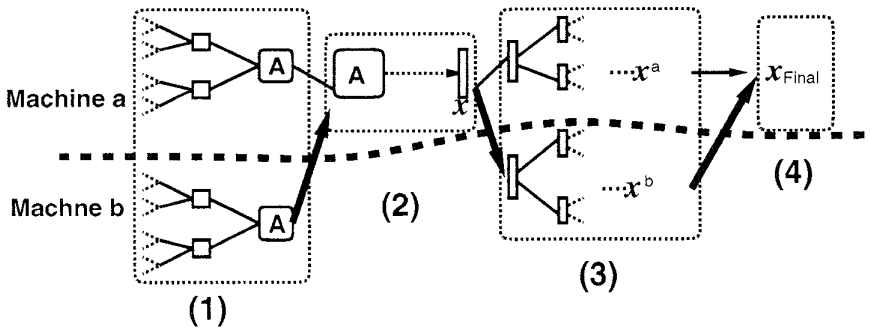

Figure 5: Implementation of the Multi-level Substructuring over Network

3 Issues in the Implementation

Generally, implementation of applications on the computers connected with the network involves several problems that comes with data transfer stage.

The main issues in data transfer can be categorized as follows:

- network latency
- the limitation of the bandwidth

Network latency can be the main issue in the applications that require numbers of synchronizations, such as iterative solvers.

The applications that use the burst data transfer and require fewer synchronizations will not be affected by the network latency. In that case, the actual performance limitation mostly depends on the bandwidth of the network, which can be much more easier to handle.

The data transfer required in the multi-level substructuring is basically the burst type of transfer and the number of the synchronization is limited in a few parts. Thus, the effect of the network latency will not be much concern compared to the iterative method.

The overall calculation process of the multi-level substructuring on parallel computers is shown in fig.5. The main calculation process can be divided into the following four stages:

1. Calculation and assembly of the coefficient matrices
2. Obtaining the solution in the highest substructure level
3. Calculation of the final solution $x_i^{(k)}$
4. Obtaining whole solution from the two parallel computers

The stage 4 consists of a series of simple substitutions only and the required processing time is small enough compared to other three stages.

Suppose all of the above processes were performed on a single parallel computer α and the processing time for the stage 1, 2,and 3 was $T_1.T_2.T_3$.

When the same calculation was performed on the two parallel computers α and β, let the calculation time for stage 1, 2, 3 as $T_1^{[\alpha|\beta]}.T_2^{[\alpha|\beta]}.T_3^{[\alpha|\beta]}$.

If the performance ratio measured in Flops was $a : 1(a \geq 1)$ and the best load balancing has been achieved, the $T_1^{[\alpha|\beta]}.T_2^{[\alpha|\beta]}.T_3^{[\alpha|\beta]}$, can be expressed as:

$$\begin{aligned} T_1^{\alpha} &= T_1^{\beta} = \frac{aT_1}{a+1} \\ T_2^{\alpha} &= T_2^{\beta} = T_2 \\ T_3^{\alpha} &= T_3^{\beta} = \frac{aT_3}{a+1} \end{aligned}$$

The data transfer between the parallel computers occurs in the three parts. The time required for each parts is defined as follows:

1. the time for the data transfer between stage 1 and 2 —— C_1
2. the time for the data transfer between stage 2 and 3 —— C_2
3. the time for the data transfer between stage 3 and 4 —— C_3

In order to achieve better performance with the two parallel computers, the following relationship should be satisfied: (fig.6)

$$\frac{aT_1}{a+1} + C_1 + T_2 + C_2 + \frac{aT_3}{a+1} + C_3 < T_1 + T_2 + T_3 \quad (5)$$

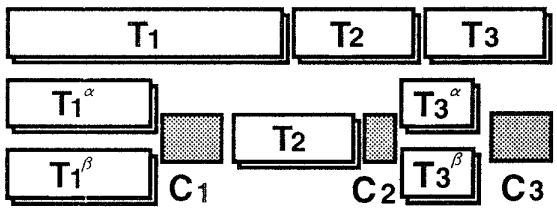

Figure 6: A Model of Total Execution Time

Transformation of Eq.(5) yields in simpler form as:

$$T_1 + T_3 > (a+1)(C_1 + C_2 + C_3) \quad (6)$$

Now we are going to examine the communication time C_1,C_2, and C_3 in more detail.

First, we assume a rectangular that is divided into finite elements as the domain of analysis.

Let the total number of the nodes in that domain n, and the number of the nodes on the boundary of the areas assigned for each parallel computer $\sqrt{n}$. In that case, data to be exchanged between the parallel computers in the stage 1 and 2 of the calculation process will be the full matrix with the size of $\sqrt{n} \times \sqrt{n}$. If each element in the matrix was expressed in double floating point numbers, and the position of that element in the assembled matrix was expressed in two integers (row and column), the approximate number of bytes to be transfered between stage 1 and 2 will be:

$$\frac{\sqrt{n} \times \sqrt{n} \times (8+4+4)}{2} = 8n \quad (7)$$

In the highest level, the solution for the nodes on the boundary of the assigned area for each computer is obtained (stage 2). This solution is passed back to the lower levels, where the solution for the whole domain is calculated (stage 3). The multiplication of the number of the nodes on the boundary $\sqrt{n}$ and the number of the bytes needed to express double floating number.

$$\sqrt{n} \times 8 \quad (8)$$

is the number of bytes transfered between the parallel computers between stage 2 and stage 3.

Data transfer from parallel computer β to α occurs between the stage 3 and stage 4. If the ratio of the number of the nodes in each assigned area is $m:1$, the size of the data to be transfered will be:

$$\frac{8n}{m+1} \quad (9)$$

As mentioned in the previous section, the number of the multiplication is $O(n^{\frac{3}{2}})$ for the problem with n nodes. If the performance ratio between α and β, which is measured in Flops, was $a:1$, m can be expressed in the following form:

$$m = a^{\frac{2}{3}} \quad (10)$$

With Eq.(10), Eq.(9) can be rewritten as:

$$\frac{8n}{a^{\frac{2}{3}}+1} \quad (11)$$

If the bandwidth of the network is B(bit/s), C_1,C_2, and C_3 will be:

$$C_1 = \frac{8n}{B} \times 8 = \frac{64n}{B} \quad (12)$$

$$C_2 = \frac{\sqrt{n} \times 8}{8} \times 8 = \frac{64\sqrt{n}}{B} \quad (13)$$

$$C_3 = \frac{8n}{a^{\frac{2}{3}}+1} \times 8 = \frac{16n}{(a^{\frac{2}{3}}+1) \times B} \quad (14)$$

4 Evaluation

We implemented a FEM solver based on multi-level substructuring on the two parallel computers (the Fujitsu AP1000 (64-cell configuration) and the NEC Cenju-3 (64-processor configuration)) connected over On-Line University network.

The domain of analysis was a rectangular domain in the area

$$0 \le x \le 1, 0 \le y \le 1,$$

which was decomposed into triangle finite elements based on 128×128 mesh. A two-dimensional Poisson differential equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \quad (15)$$

with the boundary condition

$$u_{0,y} = 100, \quad u_{x,0} = u_{x,1} = u_{1,y} = 0 \quad (16)$$

was used for the problem.

Table 1: Ratio of the execution time between Cenju and AP1000

# of nodes	Exec.time(Cenju)	Exec.time(AP)	The ratio of execution time(Cenju:AP)
4225	0.241	0.736	1:3.05
16641	1.163	3.187	1:2.73

Table 2: Execution time for the stages in calculation (# of Nodes: 16641)

stage	execution time(s)
Calculation and assembly of A_{bb}^{*} (T1)	3.08
Transfer of the coefficient matrix $A^{(k_{max-1})}$ (C1)	12.66
Calculation of $x_i^{(k_{max})}$ (T2&C2)	0.056
Calculation of x_i^{k} (T3)	0.027
Transfer of the final solution x_i^{AP} (C3)	8.27
Total	24.10

The total number of element was 32768, and the number of the node was 16641.

Fig.5 shows the overall implementation. Machine(a) and (b) in the figure corresponds to the Cenju-3 and the AP1000, respectively.

First, the domain is divided into two areas and assigned to the AP1000 and Cenju-3. Indexing table for the assigned area is created in the front end of each parallel computer. It was assumed the data necessary to start calculation (node and element information) is already stored in the local storages.

- The two parallel computers execute the first half of the calculation process until the structure level reaches k_{max-1}.
- The AP1000 sends the coefficient matrix $A^{(k_{max-1})}$ to the Cenju-3. It is assembled with the other coefficient matrix $A^{(k_{max-1})}$ created in Cenju-3 to make $A^{(k_{max})}$.
- The Cenju-3 solves Eq.(3) to obtain the solution for the highest level.
- $x_i^{(k_{max})}$ is sent back to the AP1000.
- Two parallel computers calculates x_i^{k} according to the Eq.(4) for the all substructure levels.
- The final solution obtained in the AP1000 x_i^{AP} is transfered to the Cenju-3.

Now, we discuss the performance prediction and load balancing based on the measurement of actual performance.

Previous discussions used the Flops numbers to show the difference in the performance. However, they are basically a theoretical figures and may not reflect the actual performance.

To know the performance in actual application precisely, we ran the single parallel computer version of multi-level substructuring program as a benchmark and measured the execution time. The result is shown in table 1.

Calculating the Eq.(10) using the the reversed ratio of the execution time, the proper ratio of the number of nodes can be calculated as 2.10:1 for the case of 4225 nodes and 1.95:1 for the case of 16641 nodes. The proper ratio is thought to be somewhere in between these two numbers. For the sake of simplicity, we used 2.0:1 in this implementation.

The execution time for 16641 node problem divided in the stages in the execution is shown in table 2. The lower performance compared to the execution time in table 1 can be seen.

The cause of this performance is due to the irregularity of work load and overhead in data transfer.

- An imperfect binary tree structure appears in the matrix assembly process if the number of the first level substructure is not the power of 2.

  The original version of the program was optimized for the matrix assembly process executed in the perfect binary tree structure. It worked against the imperfect binary tree in which the number of the first level substructure is not the power of 2, and resulted in slower execution time.

In addition, if the number of substructures does not match the multiple of the number of processors available the parallel efficiency is expected to be also affected.

- In the data transfer stage, the data is stored in the particular machine/processor (front-end machine for the AP1000, processor #0 for the Cenju-3) before it is sent through the network. This kind of concentration may be the cause of another bottleneck that comes before network bandwidth.

Currently we are working on the improvements in the calculation and the network routines. Once they are completed, we expect the the program to show proper performance that will match the presented model.

5 Parallel Mesh Generator

Parallelizing the Finite Element program with the large scale parallel environment mentioned in the previous sections implies the possibility of dealing with very large problems. In that case, the finite element mesh generation is expected to have much more important role. There are two reasons for this:

- Generally, the performance in the parallel Finite Element solver strongly depends on the good work load distribution. To ensure the good performance, this problem should be considered from the very beginning of the whole process.

- In large scale problems, the overhead that comes with the initial data distribution is expected to be great. Parallel mesh generation will reduce the amount of initial data and contribute to the improvement of the total throughput.

To meet these needs, we are implementing the parallel mesh generator on the Fujitsu AP1000 parallel computer. This mesh generator is also expected to be used in the heterogeneous parallel computer environment over network.

The Finite Element Method requires the mesh with proper shape in order to get a good calculation result. The Delaunay triangulation, which can be derived from the Voronoi Diagram, is known to create the good triangular mesh that suits well for that purpose. This parallel mesh generation uses the combination of the incremental method and the divide-and-conquer method to create Voronoi Diagram.

The area is first cut into strips which boundaries are parallel to the y-axis and assigned to the processors(fig.7). Each processor creates the Voronoi Diagram for the assigned area, then they are merged between processors(fig.8). Delaunay triangle mesh is created from this diagram, by connecting these nodes.

Then, the assigned area for each processor is reshaped from the long strip area to more rectangular domain to reduce the length of the boundary. The nodes and elements are redistributed accordingly (fig.9).

So far, in the 32 processor configuration, the speedup ratio of 11 was achieved for the 6000 node problem.

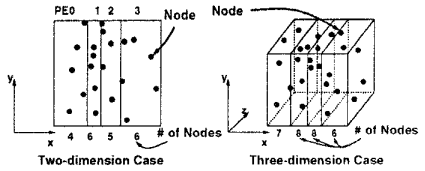

Figure 7: Node assignment to processors

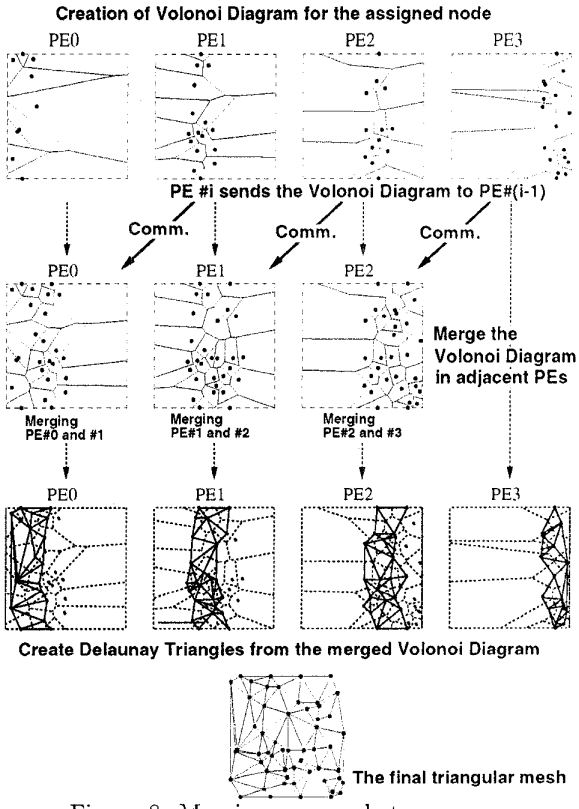

Figure 8: Merging process between processors

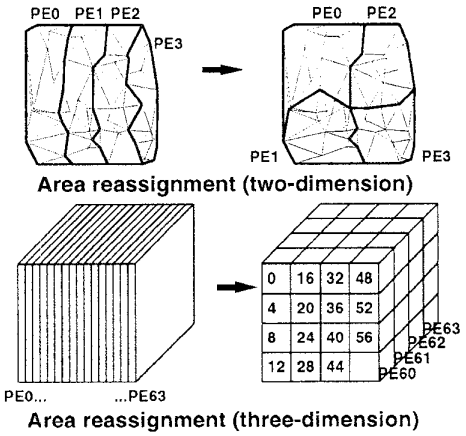

Figure 9: Reassignment of the elements

6 Conclusion

We implemented an FEM solver on the heterogeneous parallel environment using the substructure method applied in multiple levels.

Currently, the following improvement is under way to achieve better performance:

- **the research of the cause of the large communication time**
  The current data transfer rate obtained so far is far from the the expected performance of the network. The possible reason is the bottleneck caused by the concentration of the data in particular processor or front end machine. However, the definite reason is not known and further research in this issue is going to be needed.

- **the application of the parallel mesh generator**
  Heterogeneous environment version of the parallel mesh generator in is expected to provide the solution to the issues of the load balancing and the reduction of the overhead of the node and element data distribution in the initial stage. The problems with the network environment will have to be examined in the process of actual implementation.

References

[1] I.St.Doltsinis and S.Nolting: Studies on parallel processing for coupled field problems, Computer Methods in Applied Mechanics and Engineering vol.89, pp.497-521, 1991

[2] Hideo Fukumori and Yoichi Muraoka: Parallel FEM Solution Based on Substructure Method, PCW'93 Proceedings of Fujitsu Second Parallel Computing Workshop, P1-K, 1993

[3] F.J.Peters: Sparse Matrices and Substructures, Mathematisch Centrum, 1980

[4] Hideo Fukumori et al.: "Parallelization of FEM with multi-level substructure method", Proceedings of HPC'ASIA 1995, 1995