Parallel FEM Analysis for Dynamic Response of Saturated Soil Layers

Makoto KAWAMURA*, Jafril TANJUNG**

* Dr. of Eng., Professor, Dept. of Architecture and Civil Eng., Toyohashi Univeristy of Technology, 1-1 Hibarigaoka Tempaku-cho, Toyohashi 441-8580

** Graduate Student, Dept. of Mechanical and Structural System Eng., Toyohashi Univeristy of Technology, 1-1 Hibarigaoka Tempaku-cho, Toyohashi 441-8580

Analyses for dynamic response of coupling motion of a soil particle and a pore water in a saturated soil layer by 3D FEM often encounter the physical limitation of a single processor computer. In this study, at first parallel FEM formulation based on Weighted Residual Method (WRM) was derived to overcome the limitation applying Domain Decomposition Method (DDM) with an iterative solver of Conjugate Gradient (CG) method. Then a new algorithm of interprocessor communication was also developed to achieve the efficient parallel computation. Finally, the efficiency and validity of the proposed analysis were shown comparing the analytical results with experimental data.

Key Words: parallel processing, dynamic response, porous material, saturated soil layer

1. Introduction

Accurate estimation of dynamic behaviors of saturated soil layers is important to investigate seismic response of civil structures. It is not clear how much pore waters of saturated soils influence on dynamic response of the structures. When dynamic behaviors of saturated soil layers are analyzed using FEM, the model of the saturated soil as a two-phase material of soil particle and water with different mechanical properties brings more realistic results than the model of equivalent homogenous solid material[1]. 3D FEM dynamic analysis for the two-phase continuum is considered as one of the most reliable analysis. However, in this type of the analysis large number of freedom for unknowns have to be solved and the computing time by a single processor computer very often exceeds the available computer capacity.

The objective of this study is development of the parallel computation algorithm for the dynamic analysis of soil layers to overcome the problem mentioned above. In the parallel computation using a multi processor computer, it is required that imbalance of computation of each processor and communications between the multi processors are reduced as much as possible.

In Domain Decomposition Method (DDM) a whole of domain for a FEM analysis are separated into small subdomains and the computation of each subdomain is easily assigned to each processor of a multi processor computer. In DDM an iterative solver such as Conjugate Gradient (CG) method is used. CG method is effective to reduce communication time of multi processors. DDM is one of most suitable methods to satisfy the conditions for parallel computation. There are several proposed DDM algorithms in the fields of finite element analysis as reported by Tallec[2], Papadrakakis[3], and Farhat[4]. However, most of them were used for analyzing homogenous solid materials. In the previous studies on solids the formulation for DDM was derived based on variational principle, which is not applicable for a fluid. There is no application of DDM for the dynamic behavior of saturated soil layers considering granular solid and pore fluid coupling. In this paper, a derivation of a new parallel computation algorithm of 3D FEM analysis for dynamic response of saturated soil layers is presented based on DDM and using Weighted Residual Method (WRM). WRM is suitable for the solid and fluid coupling problem.

Efforts to achieve efficient interprocessors communication is also important to reduce the communication time between the processors. An efficient algorithm for interprocessor communication for hypercube system was developed in this paper.

Finally the analysis for a simple saturated soil layer was performed and the analytical results of dynamic pore water were compared with the experimental data obtained by one of authors[5]

to validate the accuracy of the developed algorithm. To examine the efficiency of the parallel computation by the proposed method, relations between number of processors and computation time were checked.

2. Fundamental Equations

Saturated soils are viewed here as the material which is composed of a granular solid and a pore fluid. The granular solid and the pore fluid have the individual mechanical properties. Assuming infinitesimal displacement the equations of motion for the saturated porous material are given as follows [6].

$$\begin{Bmatrix} \sigma_{ij,j} \\ \pi_{,i} \end{Bmatrix} - \begin{bmatrix} \rho & \rho_f \\ \rho_f & \frac{1}{f}\rho_f \end{bmatrix} \begin{Bmatrix} \ddot{u}_i \\ \ddot{w}_i \end{Bmatrix} - \begin{bmatrix} 0 & 0 \\ 0 & \frac{1}{k} \end{bmatrix} \begin{Bmatrix} \dot{u}_i \\ \dot{w}_i \end{Bmatrix} = 0 \qquad (1)$$

The properties are expressed by the following constitutive relations[1].

$$\begin{Bmatrix} \sigma_{ij} \\ \pi \end{Bmatrix} = \begin{bmatrix} C_{ijkl} + \alpha^2 m \delta_{kl} & \alpha m \delta_{kl} \\ \alpha m \delta_{kl} & m \end{bmatrix} \begin{Bmatrix} \varepsilon_{i,j} \\ \zeta \end{Bmatrix} \qquad (2)$$

where σ_{ij} are the components of the total stress tensor for the saturated porous material; π is the fluid pressure; ρ and ρ_f are the mass densities of the saturated soil and the fluid, respectively; k is the coefficient of permeability; f is the porosity; u_i and w_i are the component of displacement of the solid and the fluid, respectively. The superposed dot implies time derivative and $(\,)_{,j}$ denotes the first order derivative with respect to coordinate x_j. δ_{ij} is the Kronecker's delta; C_{ijkl} are the components of the elasticity tensor; m is the bulk modulus of the fluid; α is a measure of compressibility of solid particles representing the contact areas of the particles. ε_{ij} are the components of the solid strain; and ζ is the volume change of the fluid. In this study the dynamic behavior of saturated layer is investigated for the small strain amplitude where there is no failure in the soil layer. The mechanical property of soil skeleton assumed elastic. The relation between these strains and the displacements are written as follows.

$$\begin{Bmatrix} \varepsilon_{ij} \\ \zeta \end{Bmatrix} = \begin{Bmatrix} \frac{1}{2}(u_{i,j} + u_{j,i}) \\ w_{i,i} \end{Bmatrix} \qquad (3)$$

3. Domain Decomposition Method for Saturated Soil Layers

3.1 Domain Decomposition

The Domain Decomposition Method (DDM) is a method which refers to the substructuring techniques. Totally non-overlapping partitions split an original domain into several small disjoint subdomains. DDM converts the original problem to an interface problem[2]. With the advent of parallel processing machines DDM has become more popular for the solution of finite element analysis. A whole of domain is fictitiously divided into a set of non-overlapping subdomains. The FEM analyses of subdomains are performed under constraint of continuity of displacement among the subdomains. As an example, a domain Ω is decomposed into two subdomains $\Omega^{(1)}$ and $\Omega^{(2)}$ as shown in Fig. 1, where $\Gamma^{(12)}$ is the interface between $\Omega^{(1)}$ and $\Omega^{(2)}$. In each subdomain fundamental equation (1) and the constitutive relation (2) are satisfied. However, to enforce their continuity on their interface, the additional boundary conditions on the interface are required and written as follows.

$$\begin{Bmatrix} \lambda_{ij}^u n_j \\ \lambda_{ij}^w n_j \end{Bmatrix}^{(1)} + \begin{Bmatrix} \lambda_{ij}^u n_j \\ \lambda_{ij}^w n_j \end{Bmatrix}^{(2)} = 0 \quad on \quad \Gamma^{(12)} \qquad (4)$$

and

$$\begin{Bmatrix} u_i n_j \\ w_i n_j \end{Bmatrix}^{(1)} = \begin{Bmatrix} u_i n_j \\ w_i n_j \end{Bmatrix}^{(2)} \quad on \quad \Gamma^{(12)} \qquad (5)$$

where λ_{ij} is a traction tensor and n_j is an outer normal vector.

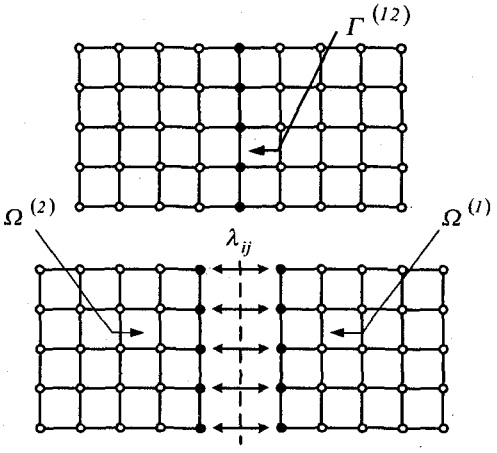

Fig.1 Decomposition into Two Subdomains

To obtain the solution for the whole domain, the displacement based weighted residual formulation for Eqs. (1) and (4), are led as follows.

$$\begin{aligned} \sum_{s=1}^{2} \Bigg[& \int_{\Omega^{(s)}} \{\delta u_i \quad \delta w_i\}^{(s)} \begin{Bmatrix} \sigma_{ij,j} \\ \pi_{,i} \end{Bmatrix}^{(s)} d\Omega^{(s)} \\ & - \int_{\Omega^{(s)}} \{\delta u_i \quad \delta w_i\}^{(s)} \left\{ \begin{bmatrix} \rho & \rho_f \\ \rho_f & \frac{1}{f}\rho_f \end{bmatrix} \begin{Bmatrix} \ddot{u}_i \\ \ddot{w}_i \end{Bmatrix} \right\}^{(s)} d\Omega^{(s)} \\ & - \int_{\Omega^{(s)}} \{\delta u_i \quad \delta w_i\}^{(s)} \left\{ \begin{bmatrix} 0 & 0 \\ 0 & \frac{1}{k} \end{bmatrix} \begin{Bmatrix} \dot{u}_i \\ \dot{w}_i \end{Bmatrix} \right\}^{(s)} d\Omega^{(s)} \Bigg] \\ & + \int_{\Gamma^{(12)}} \{\delta u_i \quad \delta w_i\}^{(12)} \left\{ \begin{Bmatrix} \lambda_{ij}^u n_j \\ \lambda_{ij}^w n_j \end{Bmatrix}^{(1)} + \begin{Bmatrix} \lambda_{ij}^u n_j \\ \lambda_{ij}^w n_j \end{Bmatrix}^{(2)} \right\} d\Gamma^{(12)} = 0 \end{aligned} \qquad (6)$$

where s denotes the number of subdomains. For the inter-subdomains constraint the following equations are given.

$$\int_{\Gamma^{(12)}} \left\{\delta\lambda_{ij}^{u} n_j \quad \delta\lambda_{ij}^{w} n_j\right\}^{(12)} \left\{ \begin{Bmatrix} u_i n_j \\ w_i n_j \end{Bmatrix}^{(1)} - \begin{Bmatrix} u_i n_j \\ w_i n_j \end{Bmatrix}^{(2)} \right\} d\Gamma^{(12)} = 0 \tag{7}$$

3.2 FEM Formulation

Applying the integration by part of Green's theorem to the first term of left hand side Eq. (6) and using the interpolating functions as the weight in the standard Galerkin method[7], the Eqs. (6) and (7) above may be written as follows.

$$\sum_{s=1}^{2}\left[\boldsymbol{M}^{(s)}\ddot{\boldsymbol{d}}^{(s)} + \boldsymbol{D}^{(s)}\dot{\boldsymbol{d}}^{(s)} + \boldsymbol{K}^{(s)}\boldsymbol{d}^{(s)}\right] = \sum_{s=1}^{2}\left[\boldsymbol{F}^{(s)} - \boldsymbol{B}^{(s)T}\boldsymbol{\lambda}\right] \tag{8a}$$

$$\sum_{s=1}^{2}\boldsymbol{B}^{(s)}\boldsymbol{d}^{(s)} = 0 \tag{8b}$$

in which

$$\boldsymbol{d}^{(s)} = \begin{Bmatrix} \boldsymbol{d}_u^{(s)} \\ \boldsymbol{d}_w^{(s)} \end{Bmatrix};\ \boldsymbol{M}^{(s)} = \begin{bmatrix} \boldsymbol{M}_u^{(s)} & \boldsymbol{M}_c^{(s)} \\ \boldsymbol{M}_c^{(s)T} & \boldsymbol{M}_w^{(s)} \end{bmatrix};$$
$$\boldsymbol{D}^{(s)} = \begin{bmatrix} \boldsymbol{D}_u^{(s)} & 0 \\ 0 & \boldsymbol{D}_w^{(s)} \end{bmatrix};\ \boldsymbol{K}^{(s)} = \begin{bmatrix} \boldsymbol{K}_u^{(s)} & \boldsymbol{K}_c^{(s)} \\ \boldsymbol{K}_c^{(s)T} & \boldsymbol{K}_w^{(s)} \end{bmatrix} \tag{9}$$

where $\boldsymbol{M}^{(s)}$, $\boldsymbol{D}^{(s)}$ and $\boldsymbol{K}^{(s)}$ are the mass, damping and stiffness matrices of each subdomain, respectively. The detailed formulations for an 8-node brick element are written in Appendix A. The subscript u and w denote the solid and fluid parts respectively, and the subscript c designates the coupled conditions. The vectors $\boldsymbol{d}_u^{(s)}$ and $\boldsymbol{d}_w^{(s)}$ are the nodal relative displacements for a solid and a fluid, respectively. The vector $\boldsymbol{\lambda}$ is composed of λ_{ij} which is the traction forces in the interface nodes. The matrix $\boldsymbol{B}^{(s)}$ is signed Boolean matrix, which localizes a subdomain quantity to the subdomain interface. If the interior degrees of freedom are numbered first and the interface ones are numbered last, the Boolean matrix $\boldsymbol{B}^{(s)}$ takes the form

$$\boldsymbol{B}^{(s)} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \tag{10}$$

where $\boldsymbol{0}$ is a null matrix and $\boldsymbol{I}$ is identity matrix.

All the matrices in Eq. (8a) are generally banded. However, when a lumped mass approach is used instead of a consistent mass, the matrix $\boldsymbol{M}_u^{(s)}$ and $\boldsymbol{M}_w^{(s)}$ become diagonal and their coupled matrix is set to the null matrix. The damping matrix for a granular solid is introduced as a form of *Rayleigh* damping[8] which is combination of the mass and stiffness matrix as written in Eq. (11).

$$\boldsymbol{D}_u^{(s)} = a_0\left(\boldsymbol{M}_u^{(s)} - f^2\boldsymbol{M}_w^{(s)}\right) + a_1\left(\boldsymbol{K}_u^{(s)} - \alpha^2\boldsymbol{K}_w^{(s)}\right) \tag{11}$$

where coefficients a_0 and a_1 are defined according to the damping ratio ζ and the natural frequency ω of the granular solid.

In the case of an earthquake loading where the input is in the form of the base acceleration $\ddot{u}_g$, the dynamic loading vector $\boldsymbol{F}^{(s)}$ is written as follows.

$$\boldsymbol{F}^{(s)} = -\begin{Bmatrix} \boldsymbol{M}_u^{(s)} & \boldsymbol{1} \\ \boldsymbol{0} & \boldsymbol{1} \end{Bmatrix}\ddot{u}_g \tag{12}$$

in which $\boldsymbol{1}$ is a unity vector.

3.3 One-step Algorithms for Solving Equation of Motion

Newmark's[9] method is most widely used direct time integration method to solve the equation of motion such as Eq. (8a). It is not necessary to apply a direct integration when mechanical property of soil particle is viewed as elastic. However in this paper the algorithm was developed expecting to be extended to non-linear conditions. Two parameter β and γ are used to determine the stability and accuracy of the algorithm. When $\beta=1/4$ and $\gamma=1/2$, the method leads to the unconditionally stable and second order accurate algorithm[10].

Applying Newmark's method and introducing subscript $n+1$ for a time step Eqs. (8a) and (8b) are written as follows.

$$\sum_{s=1}^{2}\left[\overline{\boldsymbol{K}}^{(s)}\boldsymbol{d}_{n+1}^{(s)}\right] = \sum_{s=1}^{2}\left[\overline{\boldsymbol{F}}_{n+1}^{(s)} - \boldsymbol{B}^{(s)T}\boldsymbol{\lambda}_{n+1}\right] \tag{13}$$

or

$$\sum_{s=1}^{2}\left[\boldsymbol{B}^{(s)}\overline{\boldsymbol{K}}^{(s)-1}\boldsymbol{B}^{(s)T}\right]\boldsymbol{\lambda}_{n+1} = \sum_{s=1}^{2}\left[\boldsymbol{B}^{(s)}\overline{\boldsymbol{K}}^{(s)-1}\overline{\boldsymbol{F}}_{n+1}^{(s)}\right] \tag{14}$$

$$\sum_{s=1}^{2}\boldsymbol{B}^{(s)}\boldsymbol{d}_{n+1}^{(s)} = 0 \tag{15}$$

where

$$\overline{\boldsymbol{K}}^{(s)} = \frac{4}{\Delta t^2}\boldsymbol{M}^{(s)} + \frac{2}{\Delta t}\boldsymbol{D}^{(s)} + \boldsymbol{K}^{(s)}$$

$$\overline{\boldsymbol{F}}_{n+1}^{(s)} = \boldsymbol{F}_{n+1}^{(s)} + \boldsymbol{M}^{(s)}\ddot{\tilde{\boldsymbol{d}}}_{n+1}^{(s)} + \boldsymbol{D}^{(s)}\dot{\tilde{\boldsymbol{d}}}_{n+1}^{(s)}$$

$$\dot{\tilde{\boldsymbol{d}}}_{n+1}^{(s)} = \frac{2}{\Delta t}\boldsymbol{d}_n^{(s)} + \dot{\boldsymbol{d}}_n^{(s)}$$

$$\ddot{\tilde{\boldsymbol{d}}}_{n+1}^{(s)} = \frac{4}{\Delta t^2}\boldsymbol{d}_n^{(s)} + \frac{2}{\Delta t}\dot{\boldsymbol{d}}_n^{(s)} + \ddot{\boldsymbol{d}}_n^{(s)}$$

When the values of vector $\boldsymbol{\lambda}_{n+1}$ in Eq. (14) are obtained using

CG algorithm which is explained in the next section, the relative displacement vector of each subdomain for current time step can be evaluated by the following equation.

$$\boldsymbol{d}_{n+1}^{(s)} = \overline{\boldsymbol{K}}^{(s)^{-1}}\left(\overline{\boldsymbol{F}}_{n+1}^{(s)} - \boldsymbol{B}^{(s)^T}\boldsymbol{\lambda}_{n+1}\right) \tag{16}$$

Using the displacement in Eq. (16), the velocity and acceleration vector of each subdomain for current time step is calculated as follows.

$$\dot{\boldsymbol{d}}_{n+1}^{(s)} = \frac{2}{\Delta t}\boldsymbol{d}_{n+1}^{(s)} - \dot{\tilde{\boldsymbol{d}}}_{n+1}^{(s)} \tag{17a}$$

$$\ddot{\boldsymbol{d}}_{n+1}^{(s)} = \frac{4}{\Delta t^2}\boldsymbol{d}_{n+1}^{(s)} - \ddot{\tilde{\boldsymbol{d}}}_{n+1}^{(s)} \tag{17b}$$

When the number of subdomains is N_s, Eqs. (14) and (15) are extended to Eqs. (18a) and (18b) for N_s subdomains.

$$\boldsymbol{A}_I\boldsymbol{\lambda}_{n+1} = \boldsymbol{P}_I \tag{18a}$$

$$\sum_{s=1}^{N_s}\boldsymbol{B}^{(s)}\boldsymbol{d}_{n+1}^{(s)} = 0 \tag{18b}$$

where

$$\mathbf{A}_I = \sum_{s=1}^{N_s}\boldsymbol{B}^{(s)}\overline{\boldsymbol{K}}^{(s)^{-1}}\boldsymbol{B}^{(s)^T} \tag{19a}$$

$$\boldsymbol{P}_I = \sum_{s=1}^{N_s}\boldsymbol{B}^{(s)}\overline{\boldsymbol{K}}^{(s)^{-1}}\overline{\boldsymbol{F}}_{n+1}^{(s)} \tag{19b}$$

3.4 CG Algorithms

Since $\boldsymbol{A}_I$ is a symmetrical positive definite matrix, CG method which is an iterative solver, is most suitable to compute the unique solution for Eqs. (18a) and (18b) in a parallel machine. The idea of CG method is explained in the books for the least-square optimization, written by Gill[11] and Press[12].

In the CG method, vector $\boldsymbol{\lambda}$ is obtained as unique solution of the following minimization problem. In this section, subscript of $\boldsymbol{\lambda}_{n+1}$ is temporary discarded for simplicity.

$$\min_{\boldsymbol{\lambda}}\left\{\frac{1}{2}\boldsymbol{\lambda}^T\boldsymbol{A}_I\boldsymbol{\lambda} - \boldsymbol{P}_I^T\boldsymbol{\lambda}\right\} \tag{20}$$

Suppose a non-singular matrix $\boldsymbol{S}$ satisfies the relation in Eq. (21).

$$\boldsymbol{S}^T\boldsymbol{A}_I\boldsymbol{S} = \boldsymbol{D}_I \tag{21}$$

where $\boldsymbol{D}_I$ is a diagonal matrix. If $\boldsymbol{v}$ is defined such that $\boldsymbol{\lambda}=\boldsymbol{S}\boldsymbol{v}$ then $\boldsymbol{v}$ can be determined as the solution of the following minimizing problem.

$$\min_{\boldsymbol{v}}\left\{\frac{1}{2}\boldsymbol{v}^T\boldsymbol{D}_I\boldsymbol{v} - \boldsymbol{P}_I^T\boldsymbol{S}\boldsymbol{v}\right\} \tag{22}$$

$\boldsymbol{S}$ and $\mathbf{v}$ are defined in Eqs. (23a) and (23b).

$$\boldsymbol{S} = \begin{bmatrix}\boldsymbol{s}_0 & \boldsymbol{s}_1 & \cdots & \boldsymbol{s}_k & \cdots & \boldsymbol{s}_{N_k+1}\end{bmatrix} \tag{23a}$$

$$\boldsymbol{v} = \begin{bmatrix}v_0 & v_1 & \cdots & v_k & \cdots & v_{N_k+1}\end{bmatrix} \tag{23b}$$

k is number of iteration and N_k is the number of iteration to converge. The column vector $\boldsymbol{s}_k = \begin{pmatrix}s_{1k} & s_{2k} & \cdots & s_{N_bk}\end{pmatrix}^T$ is the search direction vector of the Conjugate Gradient for the iteration of k, where N_b is the number of degrees of freedom of interface nodes. Vector $\boldsymbol{\lambda}$ is calculated in Eq. (24).

$$\boldsymbol{\lambda} = \sum_{k=0}^{N_k+1}v_k\boldsymbol{s}_k \tag{24}$$

In general the matrix $\boldsymbol{S}$ is not known *a priori* but it is possible to compute the columns of $\boldsymbol{S}$ sequentially. Once $\boldsymbol{s}_0$ is known then $\boldsymbol{v}$ can be determined in CG algorithm. The CG algorithm for solving Eq. (18) goes as follows. Initial values $\boldsymbol{\lambda}_0$ and $\boldsymbol{r}_0$ are given by Eqs. (25a) and (25b).

$$\boldsymbol{\lambda}_0 = \boldsymbol{0} \tag{25a}$$

$$\boldsymbol{r}_0 = \boldsymbol{P}_I \tag{25b}$$

The following calculations are iterated until convergence is confirmed.

$$\begin{aligned}
&Prec.\ \ z_{k-1} = \overline{\boldsymbol{A}}_I^{-1}\boldsymbol{r}_{k-1}\\
&\qquad \zeta_k = \boldsymbol{z}_{k-1}^T\boldsymbol{r}_{k-1}/\boldsymbol{z}_{k-2}^T\boldsymbol{r}_{k-2} \quad (\zeta_1 = 0)\\
&\qquad \boldsymbol{s}_k = \boldsymbol{z}_{k-1} + \zeta_k\boldsymbol{s}_{k-1} \quad (\boldsymbol{s}_1 = \boldsymbol{z}_0)\\
&\qquad v_k = \boldsymbol{z}_{k-1}^T\boldsymbol{s}_{k-1}/\boldsymbol{s}_k^T\boldsymbol{A}_I\boldsymbol{s}_k\\
&\qquad \boldsymbol{\lambda}_k = \boldsymbol{\lambda}_{k-1} + v_k\boldsymbol{s}_k\\
&\qquad \boldsymbol{r}_k = \boldsymbol{r}_{k-1} - v_k\boldsymbol{A}_I\boldsymbol{s}_k
\end{aligned} \tag{25c}$$

A preconditioner $\overline{\boldsymbol{A}}_I^{-1}$ in equation (25c) is used to enhance the convergence rate and given in Eq. (26)[4].

$$\overline{\boldsymbol{A}}_I^{-1} = \sum_{s=1}^{N_s}\begin{bmatrix}\boldsymbol{0} & \boldsymbol{0}\\ \boldsymbol{0} & \overline{\boldsymbol{K}}_{bb}^{(s)}\end{bmatrix} \tag{26}$$

Subscript *bb* in Eq. (26) designates the interface nodes.

3.5 Reorthogonalization Procedures

When the linear dynamic problem, such as Eq. (18) is solved, the different equivalent loading vector has to be evaluated on each time step while the equivalent stiffness matrix remains constant. This problem is known as repeated right-hand side (RHS) problem. The direct solver possesses a clear advantage over the iterative method since the computation of the equivalent stiffness matrix is not repeated and only one forward and one backward substitution is required for each time step. On the other hand the iterative solver must restart from every scratch and repeat iterations for every time step. To overcome the drawback of the

iterative solver, Farhat[13] has proposed a method based on the CG method incorporating with a reorthogonalization procedure. This method basically formulates the overall problem as a series of consecutive minimization problem over A_I-orthogonal and supplementary subspaces, and tailors the CG method via projection of the new interface problem onto an agglomerated Krylov space associated with previous right-hand sides. To improve this solution with an accelerated CG algorithm, all search directions are orthogonalized with respect to the Krylov subspaces generated by previous right-hand sides. This search direction modification is written as follows[13].

$$\bar{s}_{k+1} = s_{k+1} - \sum_{i=1}^{k} \frac{s_i^T A_I s_{k+1}}{s_i^T A_I s_i} s_i \tag{27}$$

As the starting point for defining vector λ_0 can be evaluated by Eq. (28).

$$\lambda_0 = \bar{S} \frac{\bar{S}^T P_{I_i}}{S^T A_I \bar{S}} \quad i = 1,2,\cdots \tag{28}$$

where $\bar{S}$ is the rectangular matrix of the orthogonalized of the previous search direction S. $\bar{s}_k$ is used only to calculated λ_0.

4. Parallel Computing Implementation

For efficiency of parallel implementation of the FEM system, the two conditions below should be always considered. First, the load imbalance among processors should be minimized as possible, and secondarily the ratio of the computation to the interprocessor communication should be maximized as possible[14,15]. The former is achieved by assigning a nearly equal number of elements to each subdomain. This condition can be done manually or by using automatic mesh partition software. The latter point depends on how the sequences of algorithms work in parallel manner.

In order to achieve the second condition above, the proposed DDM algorithms in the foregoing section were implemented as a Single Program Multiple Data (SPMD) programming model. The computer code was developed using FORTRAN-77 for Cray Origin/2000 multi processor machine. By this programming model, each processor executes the same code asynchronously without communicating each other. The synchronization takes place only when processors need to exchange data in order to obtain the overall solution. Their inter-communication was performed in the message-passing paradigm and the PVM[16] programming libraries was used as a parallel interpreter.

In the implementation, input data as well as interface condition of each subdomain has to be defined separately. Each processor reads its own data and communicates if necessary. When the code is terminated, the final computational results are obtained in each processor independently.

4.1 Subdomain Basis Algorithm

Most parts of the algorithm can be done in subdomain basis without interprocessor communication. After each subdomain is assigned to an individual processor, the processor is responsible to read/write its own data and conducts computations independently. The independent computations include forming and assembling mass matrix $M^{(s)}$, damping matrix $D^{(s)}$, and stiffness matrix $K^{(s)}$ as well as equivalent stiffness matrix $\bar{K}^{(s)}$, for each subdomain. Also, for each time step of dynamic analysis, forming and assembling the dynamic loading vector $F^{(s)}$ and their equivalent loading $\bar{F}^{(s)}$ can be done in parallel.

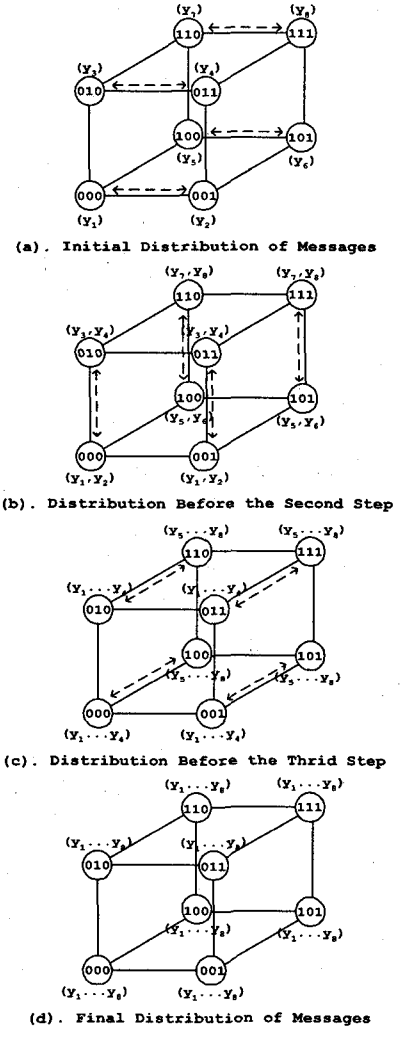

Fig. 2 Interprocessor Communication on 3-D Hypercube

Although the CG algorithm in Eq. (25) for solving Eq. (18) requires the interprocessor communication, the algorithm is still amenable to parallel manner. For example, consider matrix-vector product $A_I s_k$ in the formulation of $v_k = z_{k-1}^T r_{k-1} / s_k^T A_I s_k$. The vector y_k is evaluated as $y_k = B^{(s)} \bar{K}^{(s)^{-1}} B^{(s)^{-1}} s_k$ in each processor. In the subdomain basis algorithm using the CG method only vector y_k is exchanged

with other processor. It is effective to reduce communication. When the direct solver is used, communication time is increased to exchange matrix $\overline{K}^{(s)}$ and vector $\overline{F}^{(s)}$.

4.2 Interprocessor Communication

There are a few common patterns of interprocessor communication that are frequently used as building blocks in a variety of parallel algorithms. These patterns depend on the architecture of the parallel computer machine. Since the interprocessor communication based on hypercube can be applied for most of modern parallel computer machine, only this hypercube pattern is considered in this paper.

For the simplicity, a model for three-dimensional hypercube networking in Fig. 2 is considered. As the first step, mapping the subdomain into the three-dimensional hypercube processors such that each subdomain is assigned into the different processor separately. The neighbors in the subdomain are mapped to hypercube processors whose processor labels differ in exactly one bit position. Then, the interprocessor communication is carried out as graphically shown in Fig. 2. Assume that each vector y_k in parentheses in the figure is exchanged each other. The processors communicate in pairs when the binary number of the checked bit is different. The communication starts from the lowest dimension of the hypercube and then proceeds along successively higher dimension. At the termination of communication procedures, each processor holds the same overall result of vector y_k. A simple computer code for the algorithm is shown in appendix B.

5. Results and Discussions

5.1 Validation of Parallel Computation

For validating this parallel computational method, the experimental model based on work of Kawamura[9] has been chosen and compared with the analytical results. The model was made of saturated Toyoura fine sand and was shaken on a shaking table in sinusoidal motion with the amplitude of acceleration *300* gals and frequency of *3* Hz. The dimensions of model are *200* cm long, *100* cm wide and *56* cm high.

Table 1 Material Properties of Model

G (gf/cm^2)	ν	ρ (gf/cm^3)	m (gf/cm^2)	f	k (cm/sec)
1.5e+5	0.33	1.61	2.1e7	0.392	0.012

For the FEM analysis, the model was divided into *2640* elements and *3289* nodes with more than *17000* degrees of freedoms. The FEM mesh above then was partitioned into eight subdomains, as shown in Fig. 3 and solved by eight processors. The material properties of model are shown in Table 1. Damping ratio of granular material is assumed *10%*. The coefficient of α was taken as *1*. A time step Δt=*0.01* second was used. The responses of pore water pressures were picked up on several nodes in the left side of the model which is the vertical boundary orthogonal to the vibrating direction. A good agreement has been obtained when these responses are plotted and compared with their experimental results as shown in Fig. 4. When a saturated sand layer was vibrated in the experiment, there were frictions between the soil and the container surfaces. In the FEM analysis these frictions are not taken into account. Therefore the numerical result does not coincide exactly with the experimental result shown in Fig. 4.

In the experiment resultant force of the dynamic total stresses including effective stresses, which were acting on the wall, was 184.5 kN when the horizontal acceleration of the soil layer was 148 gals. The calculated resultant force for this condition is 191.3 kN. The predicted values for the pore water and soil skeletons are very close to the observed one.

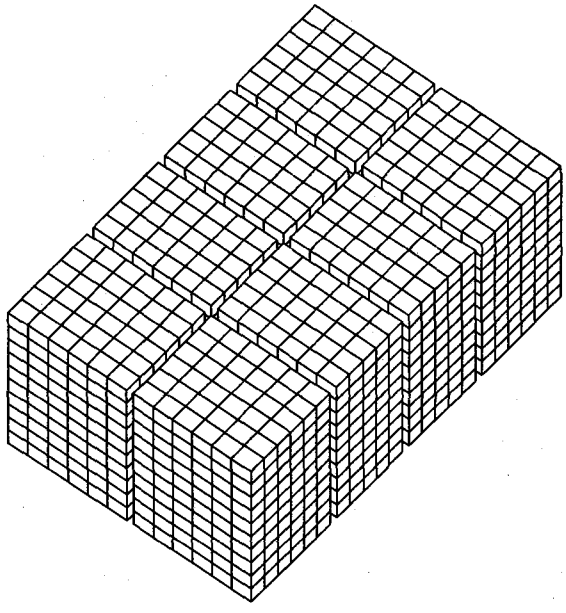

Fig. 3 Decomposition in Eight Subdomains

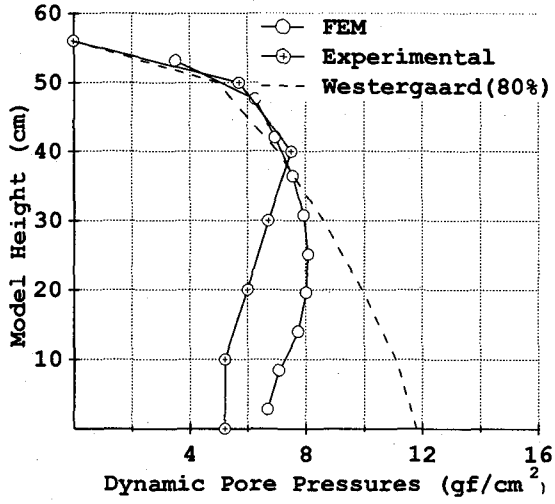

Fig. 4 A Comparison of Dynamic Pore Pressure

5.2 Performance of Parallel Computation

To evaluate the performance of this DDM algorithm, the typical saturated soil layers model mentioned above has been analyzed. The model size and the number of processors are increased by setting a constant size of the problem per processor. For each processor, the model size is kept constant as *96* elements with *900* number of degrees of freedoms. An executing computational time and the computer-memory requirement were investigated using a performance indicator. The executing computer time was recorded after the first time step of dynamic analysis finished. These values are tabulated in Table 2. The executing time in Table 2 is the CPU time for execution of command including interprocessor communication.

Table 2 Performance of Parallel Computation

Number of Processors	Problem Size	Executing time (sec.)	Memory (Mb)
1	1 x 96	12.41	4.147
2	2 x 96	13.86	4.156
4	4 x 96	14.71	4.177
8	8 x 96	14.89	4.218
16	16 x 96	16.90	4.308

The percentage of increasing of the executing computational time and the computer-memory requirement for each model scale are calculated and results are shown in Table 3. As it is expected, the parallel computation using the proposed algorithm reduces executing time as well as computer-memory requirement. Although the model is scaled in sixteen times from the original size, the executing computational time and the computer-memory requirement only increase around *27%* and *3.88%*, respectively. Proposed method is more effective than a serial method for one loading in linear case because the computations by the proposed method are conducted in the subdomain basis with less interprocessor communication. In a nonlinear case the computations should be repeated for many loading steps. Therefore the proposed method is more effective for a nonlinear case than a linear case.

Table 3 Parallel Computation Indicators

Scale of Model	Increasing of Executing Time (%)	Increasing of Memory Requirement (%)
1	0	0.00
2	10	0.21
4	16	0.71
8	17	1.71
16	27	3.88

6. Conclusions

As the results of this study followings are made clear.

1) The formulation for 3D FEM analysis of dynamic behavior of saturated soil layers with coupling solid and fluid, has been derived based on Domain Decomposition Method which is suitable for parallel computation. This formulation is derived using Weighted Residual Method. The solution in this paper has been successfully applied to the case of elastic condition which corresponds with small strain amplitude of soils. This algorithm should be more effective for the case of non-linear behavior corresponding with large strain amplitude.
2) Conjugate Gradient Method was used as the iterative solver in the Domain Decomposition derivation. Reorthogonalization procedure was applied to reduce the drawback of the iterative solver for right hand side problems.
3) To achieve efficient parallel computation the computer algorithm was developed based on the subdomains and hypercube interprocessor communication. The characteristics of the developed algorithm is that the paradigm is easy to understand what the computer machine is asked to do and that the method is applicable to most of parallel machine from Massively Parallel Processors (MPP) machines to assembles of connected several Personal Computer (PCs).
4) A simple model for a saturated soil layers was analyzed using the proposed algorithm. The analytical results of the pore water pressure were compared with the experiment data. The good agreement between the analysis and the experiment proves the accuracy of the proposed algorithm. The efficiency of parallel computation was confirmed by checking the relation between the computing time and the number of processors in the multi processor computer.

Appendix A Matrices Formulations

Detail formulation of the matrices in Eq. (9) for a three-dimensional 8-node brick element is presented in this appendix. These formulations were done as conventional FEM and can be found in several literatures[7,10]. These matrices are defined as integration of the overall of the element as shown in the following expressions.

$$\boldsymbol{K} = \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} \overline{\boldsymbol{B}}^T \overline{\boldsymbol{C}}\overline{\boldsymbol{B}} |\boldsymbol{J}| \, dr\, ds\, dt$$

$$\boldsymbol{D}_w = \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} \frac{1}{k} \boldsymbol{H}^T \boldsymbol{H} |\boldsymbol{J}| \, dr\, ds\, dt$$

$$\boldsymbol{M}_u = \rho \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} \boldsymbol{H}^T \boldsymbol{H} |\boldsymbol{J}| \, dr\, ds\, dt$$

$$\boldsymbol{M}_w = \frac{1}{f} \rho \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} \boldsymbol{H}^T \boldsymbol{H} \, dr\, ds\, dt$$

where r, s and t are the direction of the local axes of element, $|\boldsymbol{J}|$ is the determinant of Jacobian matrix and $\boldsymbol{H}$ is interpolating function matrix. While the constitutive matrix $\overline{\boldsymbol{C}}$ and the strain-displacement matrix $\overline{\boldsymbol{B}}$ are defined as follows.

$$\overline{\boldsymbol{C}} = \begin{bmatrix} c_1 & c_2 & c_2 & 0 & 0 & 0 & \alpha m \\ c_2 & c_1 & c_2 & 0 & 0 & 0 & \alpha m \\ c_2 & c_2 & c_1 & 0 & 0 & 0 & \alpha m \\ 0 & 0 & 0 & G & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & G & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & G & 0 \\ \alpha m & \alpha m & \alpha m & 0 & 0 & 0 & m \end{bmatrix}$$

in which

$$c_1 = \kappa + \frac{4G}{3} + \alpha^2 m$$

$$c_2 = \kappa - \frac{2G}{3} + \alpha^2 m$$

where κ is the bulk modulus of granular solid material, and

$$\overline{\boldsymbol{B}} = \begin{bmatrix} \boldsymbol{B}_u & 0 \\ 0 & \boldsymbol{B}_w \end{bmatrix}$$

in which

$$\boldsymbol{B}_u = \begin{bmatrix} N_{1,x} & 0 & 0 & \cdots & 0 \\ 0 & N_{1,y} & 0 & \cdots & 0 \\ 0 & 0 & N_{1,z} & \cdots & N_{8,z} \\ N_{1,y} & N_{1,x} & 0 & \cdots & 0 \\ 0 & N_{1,z} & N_{1,y} & \cdots & N_{8,y} \\ N_{1,z} & 0 & N_{1,x} & \cdots & N_{8,x} \end{bmatrix}$$

$$\boldsymbol{B}_w = \begin{bmatrix} N_{1,x} & N_{1,y} & N_{1,z} & \cdots & N_{8,z} \end{bmatrix}$$

Appendix B Example Codes

In this appendix, a simple computer code for conducting interprocessor communication of the hypercube networking in Fig. 2 based on FORTRAN-77 is introduced. The PVM parallel libraries are used as parallel interpreter tools.

Since the computer code is built in SPMD model, each of spawning codes has an integer entry number in their group working. While the hypercube processors number is designed in binary number. On the other hands, *Task Identifier* (TID) number is used to identify the active processors in a current process. The **subroutine whoiam** is used to convert the entry numbers into their binary number, and the **subroutine findpro** for converting into active TID numbers. A variable **me** in these subroutines is designed as the entry number, a variable **imyid** is the hypercube binary number and **iam** and **ippro** are TID numbers, respectively.

To explain how the processors communicate each other, consider a hypercube processor is assigned binary number **100**, for example. The **100** binary number mean is, **1**, **0** and **0** are binary codes for dimension number three, two and one, respectively. In hypercube paradigm, the communication starts from the lowest dimension and then proceeds along successively higher dimensions. In this example, the lowest dimension is denoted in binary number **0**. For the first iteration, the processor needs to find the neighbor with different **1** bit of binary code number. According to Fig. 2, the neighbor is processor with binary number **101**. To define this neighbor by a binary number, the statements of computer code below are used. See **subroutine accrevec** for detail.

```
call equal(idimen,imyid,i2i)
if(imyid(icdim).eq.0) i2i(icdim)=1
call equal(idimen,i2i,ipartner)
call findpro(idimen,ipartner,ifpro)
ippro=ifpro
```

After the pair neighbor has been found the vector data can be packed and sending to the processor which binary number **101** as a destination. At the same moment, the current processor also receives the vector data from processor where it sending data before. These processes are handled in the statements below.

```
C++ Sending data to partner processor
      call PVMFINITSEND(PVMDATARAW,info)
      call PVMFPACK(INTEGER4,lmsg,1,1,info)
      call PVMFPACK(REAL8,rstem,lmsg,1,info)
      call PVMFSEND(ippro,6872,info)
C++ Receiving data from partner processor
      call PVMFRECV(ippro,6872,info)
      call PVMFUNPACK(INTEGER4,lmsg,1,1,info)
      call PVMFUNPACK(REAL8,rstem,lmsg,1,info)
```

Almost same processes as above also perform in processor which binary number **101**. Since the binary number in current iteration is **1**, this processor should find it pair neighbor which binary number **0**.

```
      subroutine accrevec(me,idimen,lmsg,revec,
     +                    rsvec,rstem)
      include '../include/fpvm3.h'
C-----Exchange and accumulate the real data----
      implicit real*8(a-h,o-z)
      implicit integer*4(i-n)
      real*8 revec(1),rsvec(1),rstem(1)
```

```
      integer*4 imyid(0:5),i2i(0:5),ipartner(0:5)
      integer*4 me,ippro,info,iam
C
      call whoiam(me,idimen,imyid)
C-----Initialize accumulation real vector------
      do 80 ilmsg=1,lmsg
      rstem(ilmsg)=revec(ilmsg)
   80 rsvec(ilmsg)=revec(ilmsg)
C-----Seek any active processors---------------
      do 100 icdim=0,idimen-1
C-----Find partner processor-------------------
      call equal(idimen,imyid,i2i)
      if(imyid(icdim).eq.0) i2i(icdim)=1
      call equal(idimen,i2i,ipartner)
      call findpro(idimen,ipartner,ifpro)
      ippro=ifpro
      call PVMFGETTID('porwat',me,iam)
      if(iam.ne.ippro) then
C++ Sending data to partner processor
        call PVMFINITSEND(PVMDATARAW,info)
        call PVMFPACK(INTEGER4,lmsg,1,1,info)
        call PVMFPACK(REAL8,rstem,lmsg,1,info)
        call PVMFSEND(ippro,6872,info)
C++ Receiving data from partner processor
        call PVMFRECV(ippro,6872,info)
        call PVMFUNPACK(INTEGER4,lmsg,1,1,info)
        call PVMFUNPACK(REAL8,rstem,lmsg,1,info)
      endif
C-----Find partner processor-------------------
      call equal(idimen,imyid,i2i)
      if(imyid(icdim).eq.1) i2i(icdim)=0
      call equal(idimen,i2i,ipartner)
      call findpro(idimen,ipartner,ifpro)
      ippro=ifpro
      if(iam.ne.ippro) then
C++ Sending data to partner processor
        call PVMFINITSEND(PVMDATARAW,info)
        call PVMFPACK(INTEGER4,lmsg,1,1,info)
        call PVMFPACK(REAL8,rstem,lmsg,1,info)
        call PVMFSEND(ippro,6872,info)
C++ Receiving data from partner processor
        call PVMFRECV(ippro,6872,info)
        call PVMFUNPACK(INTEGER4,lmsg,1,1,info)
        call PVMFUNPACK(REAL8,rstem,lmsg,1,info)
      endif
C-----Update accumulation vector data----------
      do 125 ilmsg=1,lmsg
        rsvec(ilmsg)=rsvec(ilmsg)+rstem(ilmsg)
        rstem(ilmsg)=rsvec(ilmsg)
  125 continue
  100 continue
      return
      end
C
      subroutine whoiam(me,idimen,imyid)
C-----Define bits number of current processors-
      implicit real*8(a-h,o-z)
      implicit integer*4(i-n)
      integer*4 imyid(0:5),ilibpr(0:63,0:5)
      common/ifm/ifmecd(2)
      common/ifd/idomain
      idtemp=idimen
C-----Construct bits number of all active
      do 100 idim=1,idimen
        ncheck=(2**idtemp)/2
        jdim=(2**idim)/2
        ipr=1
        do 110 kdim=1,jdim
          nfac=0
          icode=0
          do 120 ic=1,2*ncheck
            nfac=nfac+1
            if(nfac.gt.ncheck) then
              nfac=0
              if(icode.eq.0) then
                icode=1
              else
                icode=0
              endif
            endif
            ilibpr(ipr-1,idtemp-1)=icode
            ipr=ipr+1
  120     continue
  110   continue
        idtemp=idtemp-1
  100 continue
C-----Select bits number of current processors-
      iam=me-ifmecd(idomain)
      do 130 ipr=0,2**idimen-1
        if(iam.eq.ipr) then
          do 140 idim=0,idimen-1
  140       imyid(idim)=ilibpr(ipr,idim)
        endif
  130 continue
      return
      end
C
      subroutine findpro(idimen,inpro,ifpro)
      include '../include/fpvm3.h'
C-----Convert bits number of current processor-
C-----to its ID number-------------------------
      implicit real*8(a-h,o-z)
      implicit integer*4(i-n)
      integer*4 inpro(0:5),ilibpr(0:63,0:5)
      integer*4 isum(0:63),idel(0:63,0:5)
      integer*4 iftem,ifpro
      common/ifm/ifmecd(2)
      common/ifd/idomain
      idtemp=idimen
C-----Construct bits number of all active
C-----processors
      do 100 idim=1,idimen
        ncheck=(2**idtemp)/2
        jdim=(2**idim)/2
        ipr=1
        do 110 kdim=1,jdim
          nfac=0
          icode=0
          do 120 ic=1,2*ncheck
            nfac=nfac+1
            if(nfac.gt.ncheck) then
              nfac=0
              if(icode.eq.0) then
                icode=1
              else
                icode=0
              endif
            endif
            ilibpr(ipr-1,idtemp-1)=icode
            ipr=ipr+1
  120     continue
  110   continue
        idtemp=idtemp-1
  100 continue
C-----Define bits number of current processor--
      do 130 ipr=0,2**idimen-1
      isum(ipr)=0
      do 130 idim=0,idimen-1
        if(ilibpr(ipr,idim).ge.inpro(idim)) then
          idel(ipr,idim)=ilibpr(ipr,idim)-
                         inpro(idim)
        else
          idel(ipr,idim)=inpro(idim)-
                         ilibpr(ipr,idim)
        endif
        isum(ipr)=isum(ipr)+idel(ipr,idim)
        iftem=ipr
        if((isum(ipr).eq.0).and.
     +     (idim.eq.idimen-1)) goto 135
  130 continue
  135 continue
C-----Convert bits number to ID number---------
      iffind=iftem+ifmecd(idomain)
      call PVMFGETTID('porwat',iffind,ifpro)
      return
      end
```

References

1) Ghaboussi, A.M., et. al., Seismic Analysis of Earth Dam-Reservoir Systems, *J. Soil Mech. and Found. Div., ASCE.* Vol.99, No. SM10, pp. 849-862, 1973

2) Tallec, P.L. and Vidrascu, M., Solving Large Scale Structural Problems on Parallel Computers using Domain Decomposition Techniques, *Parallel Solution Methods in Computational Mechanics (Ed. Papadrakakis, M.)*, John Wiley & Sons, Chichester, 1997

3) Papadrakakis, M., Domain Decomposition Techniques for Computational Structural mechanic, *Parallel Solution Methods in Computational Mechanics (Ed. Papadrakakis, M.)*, John Wiley & Sons, Chichester , 1997

4) Farhat, C. and Roux, F.X., A Method of Finite Element Tearing and Interconnecting and Its Parallel Solution Algorithm, *Int. J. for Num. Meth. Engg.*, Vol. 32, pp.1205-1227, 1991

5) Kawamura, M., *Studies on Lateral Earth Pressures on Retaining Walls during Earthquakes and Heavy Rainfalls (in Japanese)*, Dissertation of Dr. Eng., Nagoya University, Nagoya, 1979

6) Biot, M.A., Mechanics of Deformation and Acoustic Propagation in Porous Media, *J. Applied Physics*, Vol.33, No. 4 , pp. 1483-1498, 1962

7) Zienkiewicz, O.C. Taylor, R.L., *The Finite Element Method*, McGraw-Hill, London, 1991

8) Chopra, A. K., *Dynamic of Structures, Theory and Applications to Earthquake Engineering*, Prentice Hall, New Jersey, 1995

9) Newmark, N.M., A Method of Computational for Structural Dynamics, *J. Engg. Mech. Div., ASCE*, Vol. 85, No. EM3, Porc. Paper 2094, pp. 67-94, 1959

10)Hughes, T.J.R., *The Finite Element Method : Linear Static and Dynamic Finite Element Analysis*, Prentice Hall, New Jersey, 1987

11)Gill, P. E. and Murray, W., *Numerical Methods for Constrained Optimization*, Academic Press, New York, 1974

12)Press, W.H., et. al., *Numerical Recipes in FORTRAN*, Cambridge University Press, Melbourne, 1995

13)Farhat, C. Crivelli and Roux, F.X., Extending Substructure Based Iterative Solvers to Multiple Load and Repeated Analyses, *Comp. Meths. Mech. Enggrg.*, Vol. 119, pp.195-209, 1994

14)Baker, L., Smith, B.J., *Parallel Programming,* McGraw-Hill, New York, 1987

15). Kumar, V. et. al., *Introduction to Parallel Computing*, The Benjamin/Cummings Publishing Company, Inc., California, 1994

16)Geist, Al et. al. : PVM, *Parallel Virtual Machine, A User's Guide and Tutorial for Networked Parallel Computing*, The MIT Press, Massachusetts, 1996

(Received September 17, 1999)