Journal of Sound and Vibration (2000) **233**(1), 1–17
doi:10.1006/jsvi.1999.2790, available online at http://www.idealibrary.com on IDEAL®

SOME EXPERIENCES ON PROGRAMMING OF TIME DOMAIN BOUNDARY ELEMENT IN PARALLEL PROCESSING ENVIRONMENT

L. G. THAM AND C. K. CHU

Department of Civil Engineering, University of Hong Kong, Hong Kong

(*Received* 28 *July* 1999, *and in final form* 9 *November* 1999)

Numerical simulations of the transient dynamic response of single piles in homogenous soil under vertical loads are carried out by using the time domain boundary element method. The availability of parallel processing machine allows the programme for such analysis to be parallelized. The real time taken for computing can thus be very much reduced. This paper presents the experience gained in parallelizing the programme for the IBM 9076 Scalable Powerparallel (SP2) System. Two coarse gain and one fine grain parallelisms based on various strategies are discussed. Parallel performance results obtained in terms of speed-up and efficiency are presented to demonstrate the high level of parallelism inherent in the time domain BEM.

© 2000 Academic Press

1. INTRODUCTION

The transient response of single pile embedded in soil under dynamic loadings has long been an important research topic in geotechnical engineering. Precise analysis of the complex problem by numerical methods has been developing rapidly in the last two decades due to advances in computing power. Very robust and efficient computational schemes exist for the analysis of the problem, ranging from finite element methods (e.g., see reference [1]) to frequency domain boundary element methods [2]. Another school of thought applies the time domain boundary element approach to tackle the problem which possesses the potential to handle non-linear transient problems. It employs the Stokes solutions for the singular fundamental functions of the displacements and tractions in the Love integral equations, and solves the problems in time and space directly and simultaneously.

In this paper, a scheme for analyzing transient vertical response of single piles by the time domain method developed by Lei *et al.* [3] is adopted. The pile shaft is modelled by a number of one-dimensional finite elements while the soil domain is discretized and analyzed by a general time domain BEM formulation in cylindrical co-ordinates. By virtue of the Fourier decomposition theory, the BEM formulation is decomposed into several independent ones in accordance with the Fourier terms, and only a one-dimensional discretization is necessary along the boundaries of the soil domain. The two sets of BEM and FEM equations are then coupled into one by applying the compatibility of displacements and the equilibrium of the interactive forces between the interfaces of pile shaft and soil domain. Linear spatial and quasi-linear time interpolation functions are adopted for the tractions and field quantities.

The first parallel implementation of BEM was suggested by Simkin [4] for integral formulations of electromagnetic. A detailed implemention of BEM in a parallel environment was reported by Symm [5] through the solution of the Dirichlet problem in a circle by an array processor.

In many of the early applications, the parallelizations of BEM have been mainly at the system-equation solution phase. Various attempts have exploited the coarse grain parallelism using a vector processor on the problem of substructures in elastostatic analysis [6–8]. Calitz and du Toit [9] used an integrated array processor to affect the solution phase in an axisymmetric electromagnetic problem. Fine grain implementations for a variety of linear and quadratic element problems were firstly described by Davies [10–12] in the field of potential problems.

In order to speed up the analysis, the computer program for the analysis was parallelized and implemented on the IBM 9076 SP2 system which is a scalable parallel system based on a distributed memory MIMD message-passing architecture. This paper outlines the experiences on the parallelization of the time domain BEM. The parallelization strategies, taking into account aspects such as partitioning of the data structures and scalability, are discussed. The performance of the parallel code on the SP2 system is evaluated by using a typical example of single pile.

2. BASIC INTEGRAL FORMULATION

In the case of a pile under the action of vertical excitations, it can be shown by using Graffi's dynamic reciprocal theorem [13] that Love's [14] integral identity in a cylindrical co-ordinate system will have the form [3].

$$C_{\alpha\beta}\{u^P_{\theta_0}\} = \int_\tau \int_\Gamma A^{\mathrm{T}}(\mathbf{P}) U_{ij} A(\mathbf{Q}) \{s^Q_{\theta_0}\}\, \mathrm{d}\Gamma\, \mathrm{d}\tau - \int_\tau \int_\Gamma A^{\mathrm{T}}(\mathbf{P}) S_{ij} A(\mathbf{Q}) \{u^Q_{\theta_0}\}\, \mathrm{d}\Gamma\, \mathrm{d}\tau \qquad (1)$$

in which $i, j = x, y, z$, and Γ is the surface of the domain. Here the body force is assumed to be zero and the layer is initially at rest. **P** and **Q** are the position vectors for the receiver and source points, and $C_{ij} = \delta_{ij} - \gamma_{ij}$ (δ_{ij} is the Kronecker delta and γ_{ij} is the discontinuity term). Also, U_{ij} and S_{ij} are the fundamental displacement and traction singular solutions in Cartesian co-ordinates, respectively, u_β and s_β are the Fourier components, in circumferential direction, of the displacement and traction vectors of the source point **Q** on the surface of the domain at time τ, A(**P**) and A(**Q**) are the vector transformation matrices for **P** and **Q**.

In the analysis, the total period T is discretized into N constant time steps Δt, that is, $T = N\,\Delta t$. The field quantities (displacement and traction) between are interpolated by quadratic time interpolation functions. The half-space is also discretized into a number of nodal points (lines) along the spatial direction. Summing up the influence of the traction of each element on point **P**, and letting **P** take up the position of every boundary node, the integral equation of layer k is

$$\sum_{n=1}^{N} [\bar{H}_0]_k^{N,n} \{u_0\}_k^{n-1} + \sum_{n=1}^{N} [\bar{H}_2]_k^{N,n} \{u_0\}_k^{n}$$
$$= \sum_{n=1}^{N} [\bar{G}_0]_k^{N,n} \{s_0\}_k^{n-1} + \sum_{n=1}^{N} [\bar{G}_2]_k^{N,n} \{s_0\}_k^{n}. \qquad (2)$$

Noting the linear translation property in the time domain of the fundamental solutions [15] and the continuity condition of the displacement and traction vectors within each time step, equation (2) can be rewritten as

$$\sum_{n=1}^{N} [H]_k^{N-n+1,1} \{u\}_k^n = \sum_{n=1}^{N} [G]_k^{N-n+1,1} \{s\}_k^n. \tag{3}$$

The above equation could then be coupled with the standard FEM equation of the pile in the manner described by Lei [16] to calculate the unknown displacement (velocity and acceleration) and traction vectors long the pile shaft:

$$[K]\{U_p\}^N = \{P\}^N. \tag{4}$$

3. BASIC ALGORITHM

Consider a problem with a total of N time-steps. The structure of the original serial code is outlined below:

1. For each time-step, compute the elements of BEM coefficient matrices $[G]$ of equation (2) by numerical integration.
2. Assemble the left-hand side of equation (3) and write matrices to disk.
3. For each time-step, compute the elements of BEM coefficient matrices $[H]$ of equation (2) by numerical integration.
4. Assemble the right-hand side of equation (3) and write matrices to disk.
5. Form the characteristics matrices for the pile.
6. Read the BEM matrices coefficients from disk and couple them with the pile matrices to form a system of linear equations.
7. Solve the linear equation and calculate the unknown physical quantities.

4. MEASUREMENT FOR PERFORMANCE OF PARALLEL SCHEME

The performance of a parallel scheme can be measured in terms of the two parameters, namely speed-up and efficiency. The speed-up is a measurement of the improvement gained by the parallel program with regard to a single processor. The speed-up is defined as

$$\text{Speed-up} = \frac{\text{user time for one processor}}{\text{user time for } N \text{ processors}}.$$

It is controlled by both the inherent parallelism of the application and the efficiency of the system facilities. High speed-up implies that the computational time taken will be very much reduced but it may be achieved by using a large number of processors, and therefore, the efficiency may not be high. The efficiency of the scheme should be measured as speed-up per processor, that is

$$\text{Efficiency} = \frac{\text{speed-up}}{\text{number of processors used}}.$$

In the present study, these two parameters will be used as indices for rating the performance of the studied schemes.

5. PARALLELIZATION SCHEMES

Various parallelization schemes can be used to improve the performance of a serial program for boundary element analysis. In the present study, the following three schemes have been studied:

1. Coarse grain parallelism by matrix partition.
2. Coarse grain parallelism by time-step partition.
3. Fine grain parallelism by numerical integration partition.

The details of each scheme will be described in the subsequent sections.

To compare the performance of the schemes, a typical example of a pile under axial load is used. The pertinent parameters of the examples are as follows:

Length of pile/Radius of pile = 50.
Density of pile/Density of soil = 2·65.
Elastic modulus of pile/Elastic modulus of soil = 124·4.
Velocity of shear wave (soil) = 139·5 m/s.
Velocity of compression wave (soil) = 412·5 m/s.

The Poisson ratio of soil = 0·4.

5.1. COARSE GRAIN PARALLELISM BY MATRIX PARTITION

In analyzing the computer code of the time domain boundary element, it can be easily noted that there is no communication and data I/O involved in between the routines for forming the [G] and [H] matrices. Subroutine GC (Figure 1) forms the [G] matrix using

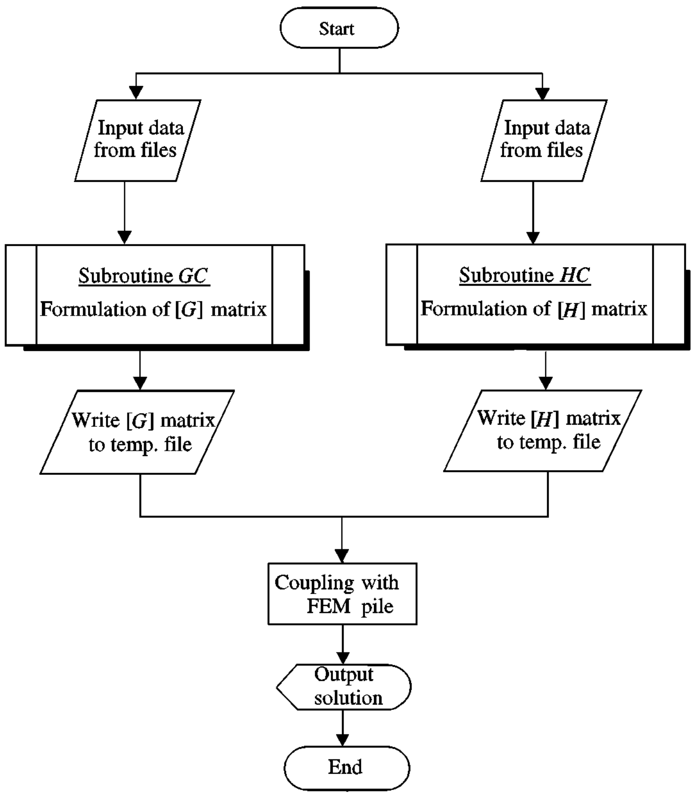

Figure 1. Flow chart for matrix partition parallelism.

Table 1

Performance of matrix partition

Case	No of spatial elements	Speed-up	Efficiency (%)
1	15	1·20	59·82
2	21	1·19	59·35
3	27	1·23	61·39
4	35	1·22	60·89
5	39	1·28	63·88
6	43	1·26	62·87

one processor (Processor 1) whereas subroutine *HC* forms the [*H*] matrix using another processor (Processor 2). Therefore, these matrices can be formed independently and the values of the coefficients can be written into the corresponding [*G*] and [*H*] data files for temporary storage until the coupling with the pile stiffness matrix.

Table 1 shows the performance of such a parallelization scheme. It is noted that the performance values are relatively low: the speed-up and efficiency are 1·2 to 1·3 and 60–63% respectively. The low parallelization efficiency is due to the inherited imbalance in work load between the subroutines [*GC*] and [*HC*]. Of course, one may consider using additional processors for forming the [*H*] matrix but it will involve re-coding and the improvement is expected to be limited.

5.2. COARSE GRAIN PARALLELISM BY TIME-STEP PARTITION

This parallelization strategy is based on a domain decomposition approach applied in the time dimension. From equation (3), it is obvious that the formation of the coefficients for [*G*] and [*H*] matrices are independent for each time-step, which for instance consists of the following numerical integration operation for time domain BEM in cylindrical co-ordinates [16]:

$$[g]^{n}_{k,m} = \int_{\Gamma_m} \int_{0}^{2\pi} \int_{\Delta t} U_{\alpha\beta} N \,\mathrm{d}\tau r \varphi_2 \,\mathrm{d}\theta \,\mathrm{d}\Gamma,$$

$$[h]^{n-1/2}_{k,m} = \int_{\Gamma_m} \int_{0}^{2\pi} \int_{\Delta t} S_{\alpha\beta} N \,\mathrm{d}\tau r \varphi_1 \,\mathrm{d}\theta \,\mathrm{d}\Gamma, \tag{5}$$

where N is the shape function for temporal integration.

The formation of these matrix coefficients are independent from each other at the time dimension and can therefore be partitioned into parallel tasks in such a way that different processors are responsible for the numerical integration of matrix coefficients for different segments of time-steps of the whole spatial mesh (for example, time-steps No. 1–5 for task 1, No. 6–9 for task 2). The arrangement is independent of the number of processors available for the problem and scalability is thus guaranteed. The diagrammatic scheme of this parallel strategy is shown in Figure 2.

It is noted that the formation of [*G*] and [*H*] matrices at different time-steps has different work loads, causing an overall load imbalance to the time-step partition parallelism when the time-steps are distributed uniformly among the processors. A typical load distribution pattern is shown in Figure 3.

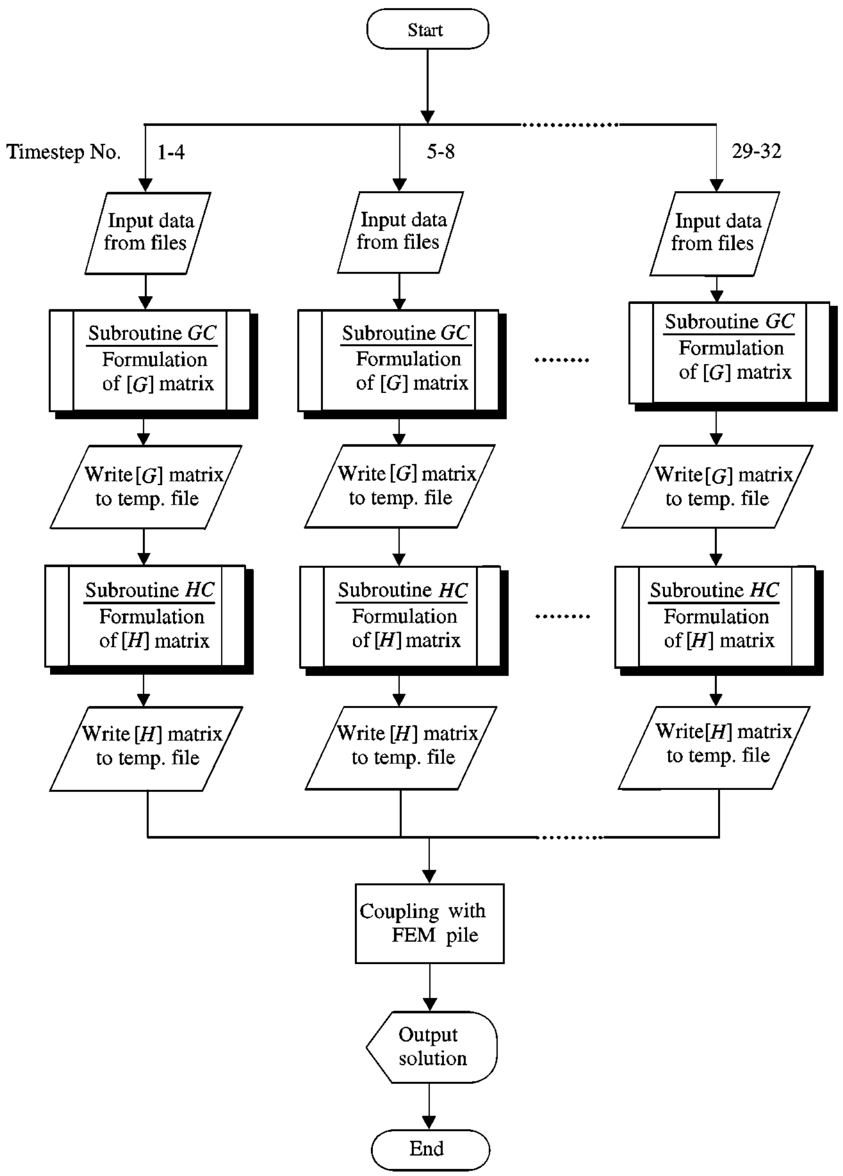

Figure 2. Flow chart for time step partition parallelism.

The reason for the load imbalance is that at the temporal integration during the formation of each matrix coefficient, the inequality condition

$$(N - n - 1)\, C_s \Delta t < r < (N - n + 1)\, C_p \Delta t \tag{6}$$

must be satisified first in order for the spatial numerical integration to be carried out. Otherwise, the trivial value of zero will be taken for the matrix coefficient and the time-consuming numerical integration procedures will be bypassed. Hence, some of the matrix coefficient formation will take longer computer time to complete. Moreover, the problem is complicated by the fact that there are 36 different cases of temporal integration

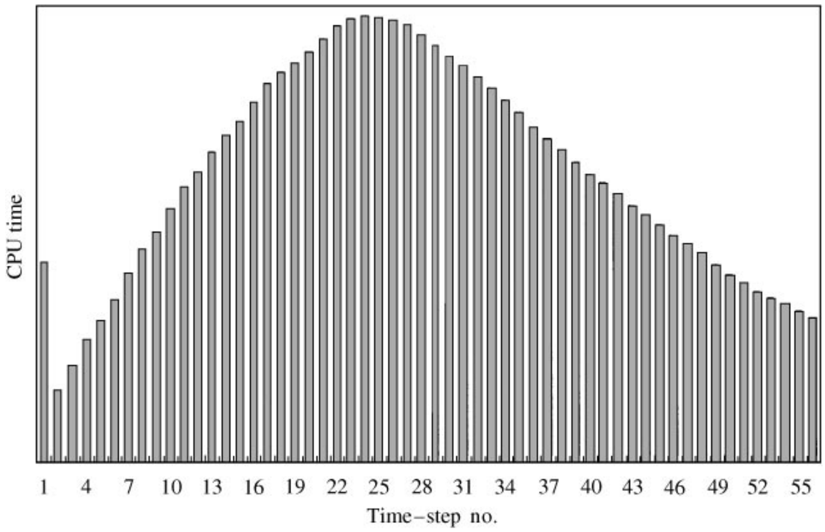

Figure 3. Load distribution among time-steps.

formulae for use in the numerical integration process and each yields a different computational work load [15].

Since the geometry of the boundary element mesh and the position of the nodes are unique in each analytical case, it is almost impossible to predict the distribution of numerical integration to be carried out for a particular combination of physical parameter and spatial boundary element mesh by any analytical formulae. The only possible way to obtain the load pattern is by actually checking node by node for each time-step to differentiate which cases a particular temporal integration formula would fall into [16]. This can be done automatically by a separate computer program or a subroutine within the original program. As no actual numerical integration is being carried out, the computation cost of this checking procedure is insignificant to the whole analysis. With the total number of each temporal integration case for the whole job recorded, the overall work load pattern can be simulated by multiplying the number of cases with the corresponding number of arithmetic operations to be carried out under the cases. This predicted load pattern in the form of a calculated number of arithmetic operations at each time-step can be used as a basis for the allocation of time-steps to each processor in order to achieve a balanced parallel computational analysis. It has been found that there are basically four types of load distribution patterns under the combinations of different boundary conditions (Figure 4).

To improve the performance of the program, a load-balancing subroutine is provided at the program structure before commencement of the BEM formulation. The target is to allocate time-steps to each processor such that the total number of arithmetic operations allocated to a processor is the closest to the average number of arithmetic operations for each processor. Starting from the first processor, the time-steps will be allocated to each processor either in ascending or descending order continuously until that target average load value has been reached. Then the next time-step will be shifted to the next processor and a similar procedure will be repeated until all the time-steps have been allocated. Whether the procedures are carried out in an ascending or descending order depends on where the predicted peak loading is located at the earlier or latter time-steps respectively. The purpose of this arrangement is to deal firstly with heaviest work load time-steps when the flexibility for allocation is greatest.

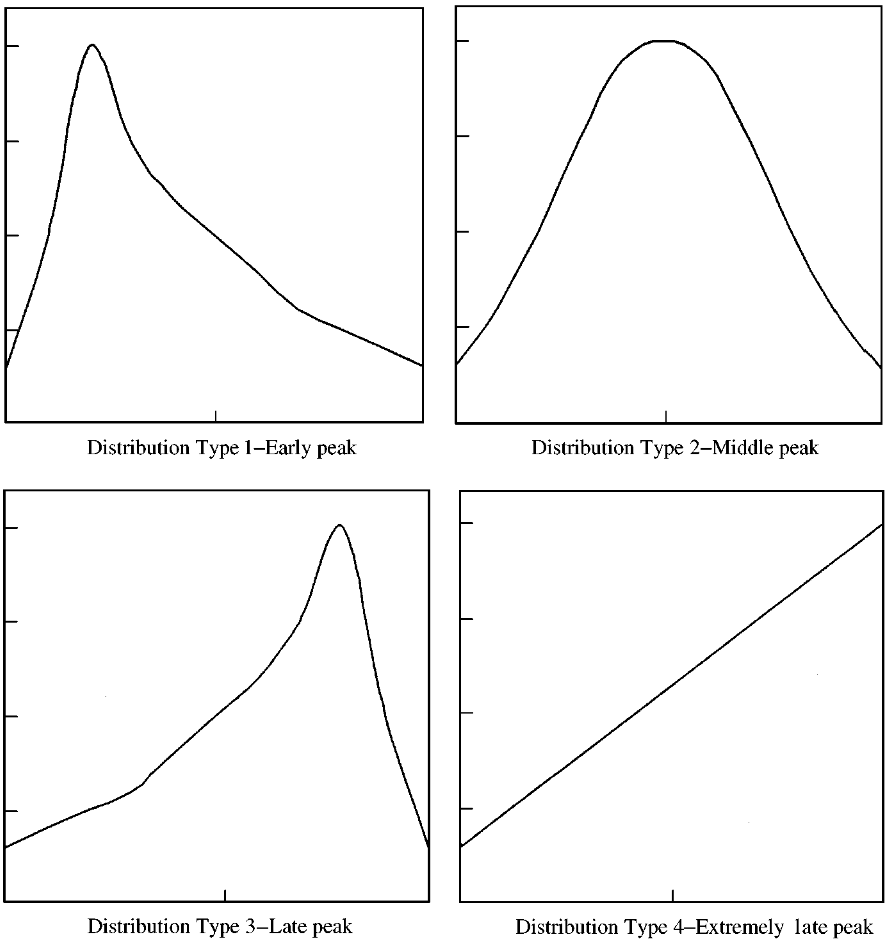

Figure 4. Load distribution patterns.

The parallel programs for both with and without load balancing were run on different combinations of physical parameters and spatial BEM meshes. The resulting parallel performance measurements of the program for a 56-time-step test job are listed in Tables 2 and 3.

It can be seen from the figures in the above tables that when the number of processors is small compared with the number of time-steps for the analysis, the parallel efficiency improves significantly after the application of load balancing.

It is also noted that there is a general tendency for the parallel efficiency to decrease as the number of processors increases owing to a significant degree of load imbalance in the time partitioning. When the number of processors/the number of time-steps ratio is small, near perfect load balancing could be achieved which results in a high parallel efficiency. However, as the number of parallel processors increases, the target number of arithmetic operations allocated to each processor — the selection criteria for load balancing — becomes smaller and smaller, which finally approaches the values of number of arithmetical operations for one time step. Since the number of time-steps allocated to each processor can never be less than unity as the time-steps cannot be divided any further, the flexibility of load balancing is greatly reduced as the number of processors

Table 2

Measured speed-up for different numbers of processors

Number of processors	Without load balancing				After load balancing			
	Type 1	Type 2	Type 3	Type 4	Type 1	Type 2	Type 3	Type 4
4	2·49	2·97	2·75	2·50	3·17	3·62	3·49	3·75
7	4·14	5·17	4·83	4·18	5·37	5·84	6·37	6·44
14	8·18	10·05	9·51	8·91	6·38	11·27	12·71	11·20
28	16·13	19·99	18·99	16·19	5·85	20·10	19·11	7·47

Table 3

Measured parallel efficiency for different numbers of processors

Number of processors	Without load balancing				After load balancing			
	Type 1 (%)	Type 2 (%)	Type 3 (%)	Type 4 (%)	Type 1 (%)	Type 2 (%)	Type 3 (%)	Type 4 (%)
4	62·23	74·20	68·64	62·55	79·16	90·58	87·16	93·73
7	59·17	73·87	68·89	59·73	76·78	83·41	90·99	92·04
14	58·45	71·80	67·95	58·47	45·56	80·52	90·45	80·02
28	57·61	71·39	67·81	57·82	20·89	71·78	67·96	26·67

increases. Nevertheless, despite this deterioration in load balancing flexibility, the use of higher number of parallel processors for the analysis still represents a substantial gain in speed-up from the original serial computation, as shown in Table 2.

In order to find the relationship between the processor/time-step ratio and the parallel efficiency for the load balancing procedures, more tests have been carried out with different combinations of number of time-steps and number of processors. For example, if the number of time-steps for the analysis is 32 and the number of processors available is 8, then the ratio equals to 0·25. A higher ratio means that the number of processors utilized for the analysis is higher and hence the flexibility for load balancing decreases. The average values of parallel performance measured are compared with the values obtained without load balancing. The results are summarized in Figure 5. When the value of the ratio is higher than 0·35, it is observed that the load balancing procedure actually has an adverse effect on the parallel performance which is lower than the cases without load balancing. Therefore, it its not recommended to apply the load balancing subroutine if the value of the processor/time-step ratio is higher than 0·35. This can be done by adding a condition line into the program to determine whether the load-balancing subroutine instead of an even distribution of time-steps should be carried out, by checking the value of processor/ time-step ratio. As such, an average minimum parallel efficiency of about 60% for the time-step partition parallelism could be maintained. Therefore, for the case of using 28 processors, the speed-up obtained using this time-step partition parallelism would be at least 16·8, that is, the parallel code can be completed in a time 16·8 times shorter than the original serial code.

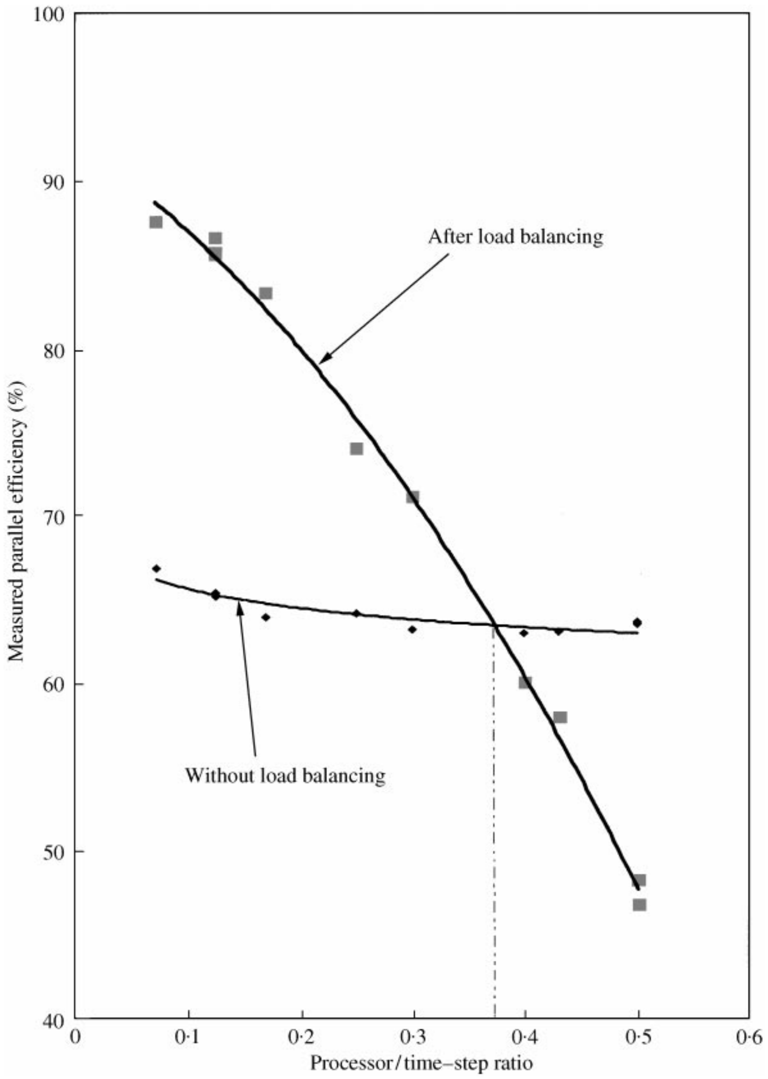

Figure 5. Parallel efficiency versus processor/time-step ratio.

5.3. FINE GRAIN PARALLELISM BY NUMERICALLY INTEGRATION PARTITION

The third approach explores the possibility of parallelization of the numerical integration processes which are commonly found in boundary element analysis. Individual matrix coefficients for the integral equation system consists of integrals which are two dimensional in space with an additional time dimension integration such as the following example [16]:

$$[g]^{n-1}_{\xi,m} = \int_0^{2\pi} \int_{-1}^{1} \int_{\Delta t} U_{\alpha\beta} N \, \mathrm{d}\tau \, \varphi_2 \, r \, | J_m | \, \mathrm{d}\eta \, \mathrm{d}\theta, \tag{7}$$

where N is the shape function for temporal interpolation.

In obtaining the values of these coefficients, the integration over time dimension is calculated analytically while the 2-D spatial integration is calculated numerically by applying the Gauss–Legendre integration formula [17]:

$$\int_{-1}^{l} \int_{-1}^{l} f(\xi, \eta)\, \mathrm{d}\xi\, \mathrm{d}\eta = \sum_{i=1}^{M} \sum_{k=1}^{N} f(\xi_i, \eta_k)\, w_i w_k, \tag{8}$$

where ξ_i and η_k are the Gaussian points, and w_i and w_k are the associated quadrature weights. $f(\xi, \eta)$ is the function to be integrated. Hence, the spatial integration in two dimensions is calculated by two summation series and this same process is repeated for every matrix coefficient. In devising the current fine grain parallel strategy, the outer summation series from $i = 1$ to M for w_i can be partitioned into parallel tasks such that each processor will calculate a part of the summation series. For best parallel efficiency and minimum load imbalance, the summation steps taken by each processor should preferably be evenly distributed and identical to each other. After all the partial summations are completed by individual processors, the results of the partial sums are transmitted to the "parent" processor for calculating the final overall sum. This final sum will be the value of the matrix coefficients and stored in the memory of the "parent" processor before writing to the $[G]$ and $[H]$ matrices files after the completion of subroutines *GC* and *HC*. During the process, synchronization is required to ensure the accurate passage of message and smooth running of the program. The procedure will be repeated for another matrix coefficient and so on until the whole $[G]$ and $[H]$ matrices have been formed. The parallel partial summations for a matrix coefficient will be carried out one by one in a sequential manner to reduce the demand for memory buffer to store the partial summation values.

Consider a 100-numerical-summation-step program running on a 4-processor parallel environment; the parallelism can be summarized in the flow chart as shown in Figure 6. Using a typical example, the speed-up and efficiency of this scheme was studied and the results are presented in Figures 7 and 8. It can be observed that the speed-up achieved for the analysis increases as the number of processors increases. In both the speed-up and efficiency charts, it is particularly interesting to note that when the value of the number of parallel processors is equal to 2, 4, 5, 10, 20 and 25, the parallel performance is remarkably higher than that of the other values. These values are factors of 100, i.e., the total number of summation steps adopted for the numerical integration of the program. Only at these values can the numerical integration be evenly distributed among the processors. For the values of number of parallel processors which are not factors of 100, some of the processors would be allocated with more summation steps while some with lesser for each numerical integration carried out during the formulation of the $[G]$ and $[H]$ matrices. The processors with fewer summation steps would waste more time on idling while waiting to send the result of the partial summations to the "parent" processor because they have to wait until every processor has completed its partial summation, due to synchronization call at the program.

As shown in Figure 9, the overhead time utilized by non-factor values of number of processors is much higher than that of the factor ones.

A comparison of the balance of work load for different values of number of parallel processors is presented in Figures 10 and 11. Figure 10 shows the load balancing chart when the program runs on 20 processors. In this case, each processor would carry out five summation steps for the numerical integrations. The total load for each processor appears to be evenly balanced with a shape of perfect circle, again due to the synchronization. The overhead time for each processor is also very low, at only about 18% of the total time. It consists of mainly the communication time between the 19 processors with the "parent" processor, No. 0, and also the idling time by the 19 processors when Processor No. 0 is

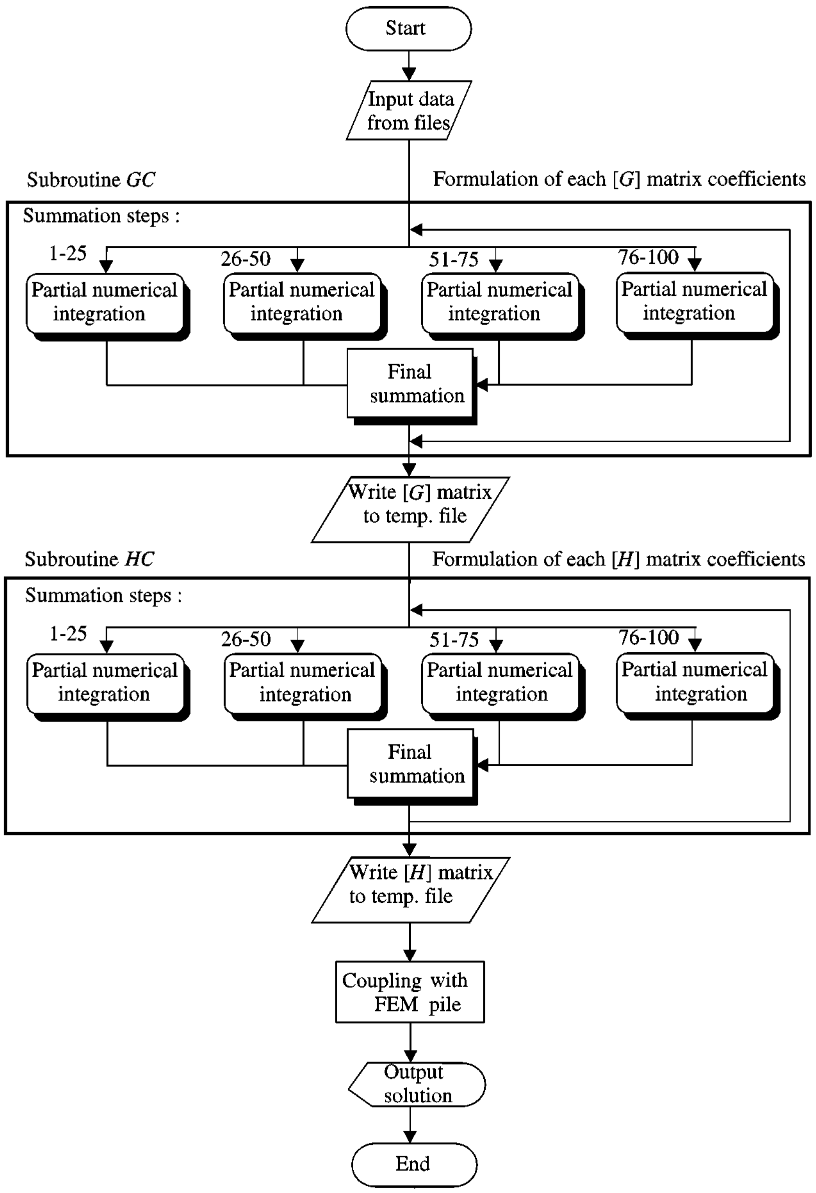
Start
Input data from files
Subroutine GC
Formulation of each [G] matrix coefficients
Summation steps :
1-25
26-50
51-75
76-100
Partial numerical integration
Final summation
Write [G] matrix to temp. file
Subroutine HC
Formulation of each [H] matrix coefficients
Summation steps :
Write [H] matrix to temp. file
Coupling with FEM pile
Output solution
End

Figure 6. Flow chart for fine grain parallelism by numerical integration partition.

carrying out activities other than numerical integrations. On the other hand, Figure 11 which shows the load balancing for 21 parallel processors is quite different from Figure 10. The overhead times for individual processors are much higher than those for 20 processors, and they also differ from each other. For a job with 21 parallel processors, the processors no. 0, 4, 8, 12 and 16 are allocated with only four summation steps in the numerical integration while the other processors are allocated with five steps. As seen from the load balancing chart, all of the processors allocated with four steps, with the exception of Processor No. 0 which is the "parent" processor, have utilized more overhead time than the other processors allocated with five steps. Although they have been allocated with less

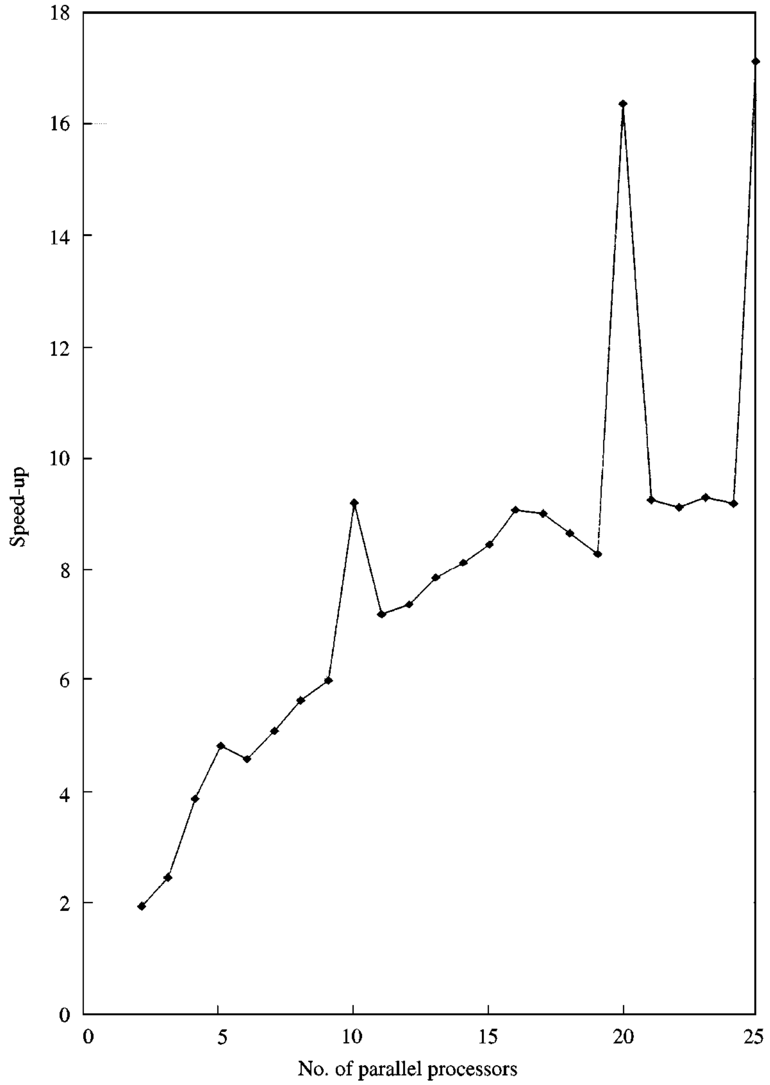

Figure 7. Speed-up versus number of parallel processors.

work load, they have spent more time on waiting to send the partial summation result than the other processors which have one more summation step to do. It has an accumulative effect on the idling time, causing the much higher values in the overhead time. Hence, it demonstrates the importance of a balanced load among processors on the overall parallel performance of the program.

Another important factor governing the amount of overhead time of a parallel job in the current parallelism is the number of processors utilized. It can be observed from Figure 7 that the parallel efficiency decreases as the number of parallel processors increases. It is obvious that as the number of processors increases, the amount of inter-processor communication increases. There are two major effects: (i) increasing the number of communication; (ii) elongating the path of communication. This is a common disadvantage for any communication-intensive parallelization scheme running on a distributed memory MIMD machine. When the number of processor is 21–24, the overhead time is more than

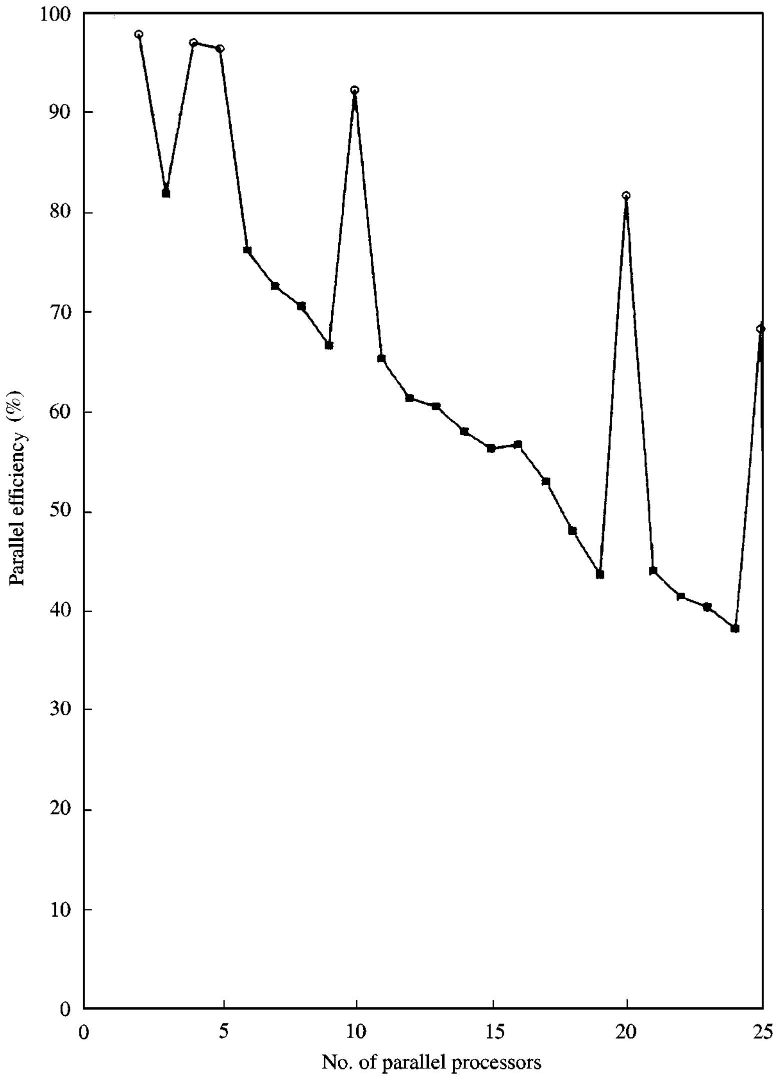

Figure 8. Parallel efficiency versus number of parallel processors.

half of the total CPU time cost. But despite this, the parallel job is still more than nine times faster than the original serial job and it has fully demonstrated the power of parallel computing.

6. CONCLUSIONS

Three different parallelisms of a time domain boundary element method for the analysis of transient response of vertically loaded single piles are presented. The parallelisms are designed according to different approaches on partitioning of the original serial problem and different degrees of inter-processors communications. The performance of the parallel programs written according to the different parallelisms has been measured through the

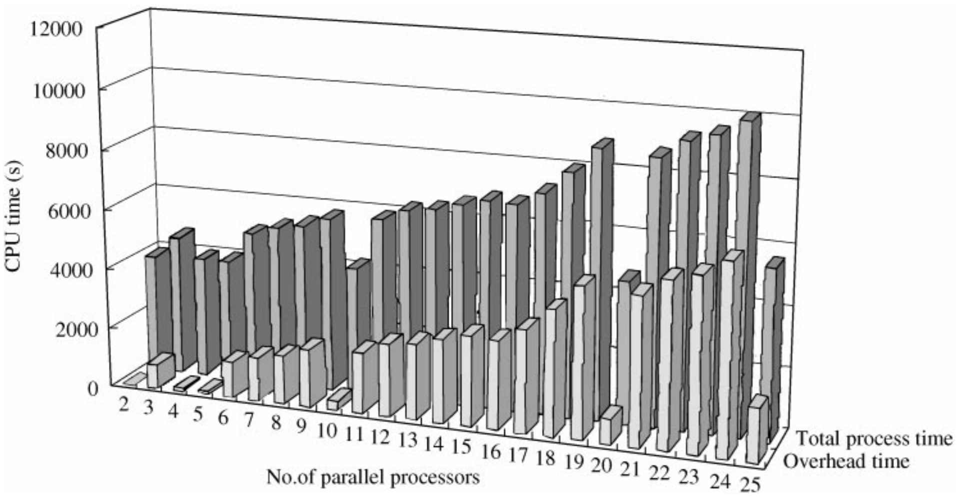

Figure 9. Breakdown of computer time utilized for different numbers of processors.

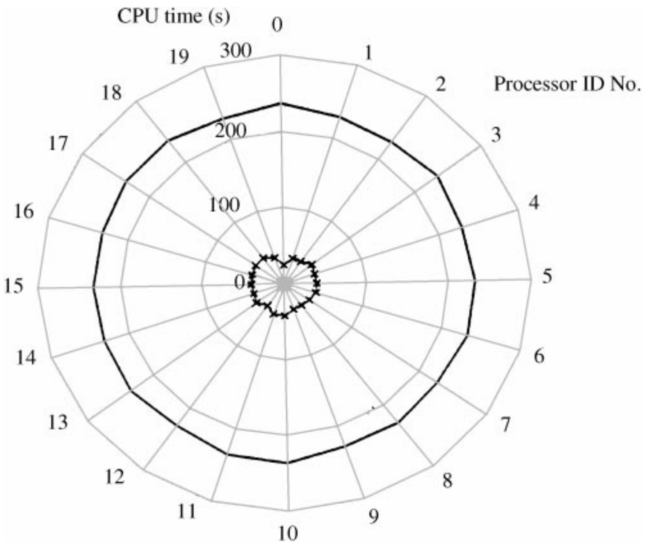

Figure 10. Load-balancing chart for 20 processors. —— Total time; —×— communication time.

implementation of the programs on the IBM 9076 SP2 parallel computer with problems of different boundary conditions.

The matrix partition parallelism has an inherent load imbalance which prohibits its parallel efficiency. The efficiency of the time-step partition parallelism is also affected by the load imbalance due to the integration at the time dimension but this can be overcome by load-balancing procedures. The load-balancing issue does not present a big problem to the numerical integration partition parallelism but its efficiency is limited by the expensive overhead costs required by the transfer of the essential data during the merging of integration. Both the matrix partition and time-step partition parallelisms are highly portable, which means that the programmes written for these parallelisms can be easily

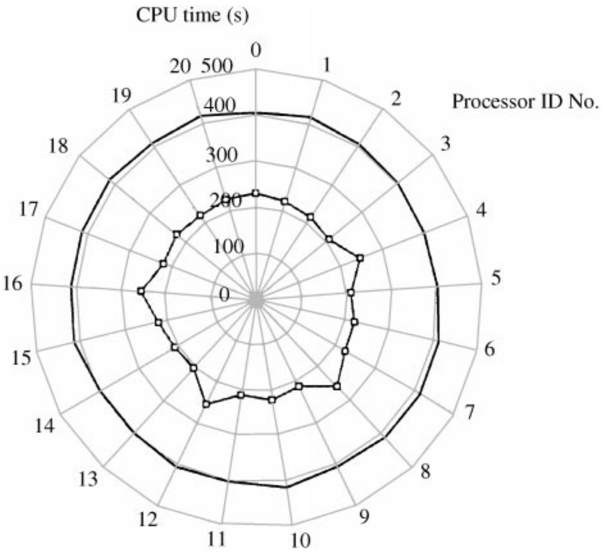

Figure 11. Load-balancing chart for 21 processors: —— Total time; —□— Communication time.

transfered from one MIMD computer platform to another. On the other hand, it would be more difficult to transfer the numerical integration partition parallelism as all the message-passing subroutines have to be modified to suit each environment.

REFERENCES

1. G. W. Blaney, E. Kausel and J. M. Roesset 1976 *Proceedings of the 2nd International Conference on Numerical Methods in Geomechanics*, 1001–1012. Dynamic stiffness of piles.
2. R. Sen, T. G. Davies and P. K. Banerjee 1985 Dynamic analysis of piles and pile groups embedded in homogeneous soils. *International Journal of Earthquake Engineering and Structural Dynamics* **13**, 53–65.
3. Z. X. Lei, Y. K. Cheung and L. G. Tham 1993 *Soil Dynamics and Earthquake Engineering* **12**, 37–49. Vertical response of single piles: transient analysis by time-domain BEM.
4. J. Simkin 1982 *IEEE Transactions on Mangnetics* MAG-**18**, 401–405. A comparison of integral and differential equation solutions for field problems.
5. G. T. Symm 1984 *Engineering Analysis* **1**, 162–165. Boundary elements on a distributed array processor.
6. D. G. Bozek, D. M. Ciarelli, K. J. Ciarelli, M. F. Hodous, R. B. Katrick and K. A. Kline 1983 *DEF Bulletin de la Direction des Etudes et Recherches*, *Serie C-Mathematique*, Vol. 1, 87–94. Vector processing applied to boundary element algorithms on the CDC Cyber-205.
7. K. A. Kline, N. K. Tsao and C. B. Friedlander 1985 *Advanced Topics in Boundary Element Analysis*, AMD-72 (T. A. Cruse, A. B. Pifko and H. Armen, editors), 257–269. New York: ASME. Parallel processing and the solution of boundary element equations.
8. J. H. Kane, B. L. K. Kumar and S. Saigal 1990 *Computational Methods in Applied Mechanics and Engineering* **79**, 219–244. An arbitrary condensing, non-condensing solution strategy for large scale, multi-zone boundary element analysis.
9. M. F. Calitz and A. G. du Toit 1988 *IEEE Transactions on Magnetics*, MAG-**24**, 427–430. CAD system for cylindrically symmetric electric devices.
10. A. J. Davies 1988 *Parallel Computing* **8**, 348–353. The boundary element method on the ICL DAP.
11. A. J. Davies 1988 *Boundary Elements X*, Vol. 3 (C. A. Brebbia, editor), 657–666. Berlin: Springer-Verlag. Quadratic isoparametric boundary elements on the ICL DAP.

12. A. J. DAVIES 1989 *CONPAR* 88 (C. R. Jesshope and K. D. Reinartz, editors), 230–237. Cambridge: Cambridge University Press. Mapping the boundary element method to the ICL DAP.
13. D. GRAFFI 1946 *Memorie della accademia delle scienze*, Series **10**, 103. Sul theorema di reciprocta nella dinamica dei corpo elas - ticiti.
14. A. E. H. LOVE 1904 *Proceedings of London Mathematical Society* **2**, 231–344. The propagation of wave motion in an isotropic elastic medium.
15. W. J. MANSU and C. A. BREBBIA 1982 *Applied Mathematical Modelling* **6**, 299–306. Numerical implementation of the boundary element method for two-dimensional transient scalar wave propagation problems.
16. Z. X. LEI 1993 *Ph.D. Thesis*, *University of Hong Kong*, *Department of Civil and Structural Engineering*. Time domain boundary element method & its applications.
17. A. H. STROUD and D. SECREST 1966 *Gaussian Quadrature Formulas*. Englewood Cliffs, NJ: Prentice-Hall.