Jurnal Kejuruteraan 16 (2004) 67-78

Finite Element Model for Crack Propagation Using Master-Workers Method

Ahmad Kamal Ariffin, Choy Hau Yan
and Mohd Jailani Mohd Nor

ABSTRACT

This paper describes an algorithm for parallel assembling of the stiffness matrix in simulation of crack propagation in distributed memory environment using master-workers method. In this algorithm, element stiffness matrix is formed by groups in each processor related by the finite element mesh. Each processor assembles a specific group of elements and no synchronization is required to avoid two or more worker processors sending the calculation result to master processor concurrently. This paper gives the speed-up rate in the simulation of crack propagation. The results indicate excellent performance and reduction of total computational time.

Keywords: Parallel processing, finite element method, adaptive crack propagation

ABSTRAK

Rencana ini menerangkan satu algoritma untuk pemasangan matriks kekukuhan selari dalam simulasi perambatan retak dalam sekitaran memori teragih menggunakan kaedah induk-pekerja. Dalam algoritma ini, matriks kekukuhan unsur dibentuk dalam setiap pemproses dihubungkan oleh jejaring. Setiap pemproses memasang sekumpulan unsur dan penyegerakan tidak diperlukan bagi mengelakkan dua atau lebih pekerja menghantar keputusan pengiraan secara serentak kepada induk. Kertas kerja ini memberikan kenaikan kadar-laju dalam simulasi perambatan retak. Keputusan menunjukkan prestasi yang sangat baik dan menurunkan masa pengiraan.

Katakunci: Pemprosesan selari, kaedah unsur terhingga, perembatan retak suai

INTRODUCTION

Users of finite element programs attempt to obtain solutions to larger problems. Many encounter major technical barriers; limitations in memory or in CPU speed, or both. One of the remedies is parallelisation. The computing time can be reduced to some extent by algorithmic changes and the radical speed increase can be reached with multiprocessor computation. The current trend in parallel processing is to connect complete computing units (processor and memory) with a high-bandwidth communications network (Watson & Noor 1996). The numerical analysis of structure is among those that were most benefited by the arrival of these parallel computers (Rezende & Paiva 1999).

Since most of the parallel computers have distributed memory, an essential code modification is necessary for porting the sequential code to

parallel version (Nikishkov & Kawka 1998). Taxonomy of architectures for parallel programming used is Single Instruction Multiple Data (*SIMD*). Each processor executes exactly the same instruction, but uses its own data. Different program can be written and operated in a single program for different processor. This method operates with the duty being divided into small individual's division and solved by processor.

In this paper, an adaptive finite element mesh analysis is used to simulate crack propagation for two-dimensional elastoplastic condition. A computer code has been developed using FORTRAN programming language for finite element analysis calculation process based on displacement formulation.

FINITE ELEMENT EQUATION AND PARALLELIZATION STRATEGY

In crack propagation simulation, the geometry of the domain changed at each step of crack propagation at the vicinity of the crack (Syifaul 2002). The smaller the finite element mesh sizes, the more accurate the finite element approximate solution. Mesh refinement process will continue until specified size achieved. Reduction in the mesh size will lead to longer computational time.

An adaptive finite element mesh is applied to analyse two-dimensional elastoplastic micro-fracture during crack propagation. An automatic remeshing is calculated at each step of crack advance based on stress error estimation of the element shape. The crack is free to propagate without predetermined path and direction. Crack tip opening angle (CTOA) is used as a criterion for crack growth, while the maximum principal stress is used as its direction.

In general, the smaller the finite element mesh, the more accurate the finite element approximate solution. However, reduction in the mesh size leads to more computational effort. Therefore it is more attractive to selectively refine the mesh in areas where the error in the approximate solution is the highest. One example of the crack propagation simulation is when a two-dimensional plate with a single crack is subjected to uniformly distributed loading on the top boundary segment of the domain as in Figure 1.

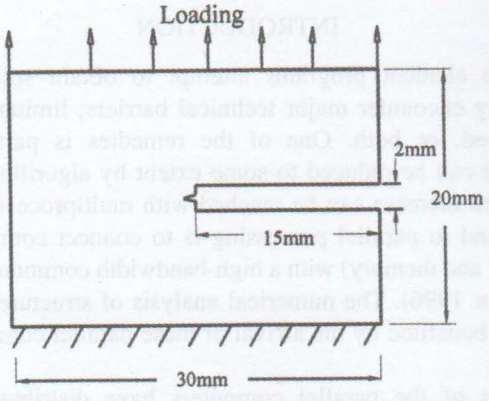


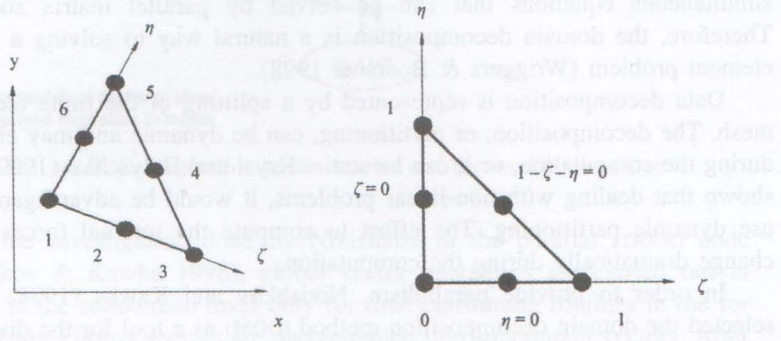
FIGURE 1. Two-dimensional plate with single crack

In the calculation, the three-noded triangle element mesh is automatically changed into a six-noded element. The mesh refining processes will mesh continuously until the optimum mesh arrangement with error permissible at the element given is obtained. The number of element and number of node are changed for each sequence. The size of element is different in whole domain area. In the area of higher stress concentration, the size of element is smaller then other place.

The finite element model in solving the crack propagation problem is based on displacement formulation. Six-noded triangular elements are used since it can fill most of the element at border. The step for modelling finite element starts by dividing the domain to small finite element, Ω_e. Composition of these elements will form a finite element domain model,

$$\Omega = \sum_{e=1}^{n} \Omega_e \tag{1}$$

which the system coordinates that is used is shown in Figure 2.



(a) Global (b) Local

FIGURE 2. Global and local coordinate system in triangular element

The displacement equation for each element can be written as follows:

$$[K_e]\{\Delta a_e\}=\{r_e\} \tag{2}$$

where $[K_e]$ is elemental stiffness matrix, $\{\Delta a_e\}$ is incremental displacement vector in boundary environment and $\{r_e\}$ is force vector not included in the analysis. Equation for elemental stiffness matrix is given by,

$$[K_e] = \Omega_e \int B_e{}^T D_{ep} B_e d\Omega \tag{3}$$

where B_e is elemental strain-displacement matrix and D_{ep} is elastoplastic material property matrix. The Gaussian technique is used to calculate the integration (Chandrupatra & Belegundhu 1997) and finally the global stiffness matrix becomes

$$[K] = \sum_{e=1}^{n} K_e \tag{4}$$

The solution for linear equation system is based on incremental iteration technique. According to the updated Lagrangian strategy, the values of displacements and stresses are used with reference to previous calculations. Incremental stress during the plasticity must fulfil the yield criterion and plasticity flow. The calculation must be calculated until the stress state returns to the stress yield surface when the plasticity level is achieved.

In the incremental iteration of the solution, the first step takes the value of the material elastic matrix, D_e. Calculation results from the first step become the input for the following step where characteristic of the material plastic elastic matrix, D_{ep} are considered into the material matrix, D. Then, for each increment, the stresses, strains, coordinates and reactions were updated according to the Lagrangian formulation.

DOMAIN DECOMPOSITION AND LOAD BALANCING

Domain decomposition is most commonly used in solving solution. The solution space is divided up among the processors and each processor solves its own piece. This method of solution often leads naturally to a set of simultaneous equations that can be solved by parallel matrix solvers. Therefore, the domain decomposition is a natural way to solving a finite element problem (Wriggers & Boersma 1998).

Data decomposition is represented by a splitting of the finite element mesh. The decomposition, or partitioning, can be dynamic and may change during the computation, or it can be static. Krysl and Belyschko (1997) has shown that dealing with non-linear problems, it would be advantageous to use dynamic partitioning. The effort to compute the internal forces may change dramatically during the computation.

In order to provide parallelism, Nikishkov and Kawka (1998) have selected the domain decomposition method (DDM) as a tool for the division into parallel tasks. According to the DDM, the finite element domain is partitioned into subdomains, and a large part of calculation is performed at the subdomain level without interprocessor data communication (Nikishkov & Kawka 1998).

Load balancing is used to distribute computations fairly across processors in order to obtain the highest possible execution speed. Load balancing can be attempted statically before the execution of any process or dynamically during the execution of the processes. Some systems may have communication delays that vary under different circumstances, and it could be difficult to incorporate variable communication delays in static load balancing. Therefore, dynamic load balancing can be used for these circumstances. In dynamic load balancing, tasks are allocated to processors during the execution of the program. The master processor holds the collection of tasks to be performed. Tasks are sent to the slave processors. When a slave completes one task, it requests another task from the master process (Wilkinson & Allen 1999). However, dynamic load balancing is a rather complicated and evolving issue, for which no simple solutions exist. Therefore, static domain decomposition is used in this work.

PARALLEL ALGORITHM WITH MASTER-WORKER COMMUNICATION

Parallelization for computer networks requires much larger messages to ameliorate the effects of large latencies (Krysl & Belyschko 1997). Master-workers model techniques configuration of a central 'master' program communication with a number of 'workers' (Geist et. al. 1997). At the beginning of calculation, the master processor receives all the input data from user. Then all the data input are broadcasts from master processor to all other processor (Jaques & Ross 1994). The system stiffness matrix is partitioned into a number of equally sized smaller domain matrices, each of which is allocated to a separate processor as in Figure 3. The domain matrices are solved independently. As soon as the calculations completed, domain matrices from each processor are then send to master processor for assembly.

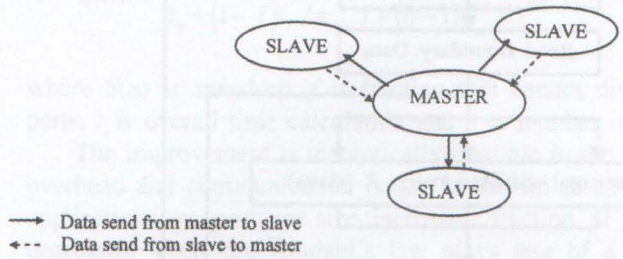


FIGURE 3. Configuration of the master and the workers in a star

In the investigation done by Nikishkov in the parallel ITAS3D code (Nikishkov & Kawka 1998), global vector operations are carried out in parallel at the subdomain level only for time consuming routines in the for subdomains. Global vectors are disassembled into subdomain vectors. After subdomain calculations, subdomain vectors are assembled into domain vectors, and modules with small consumption of computing time are run on all the processors in a serial mode. Both vectors and subdomain vectors are stored at each processor node. The disassembly operation is carried out at each processor node without data communication. A possible disadvantage is the serial part of the program worsens parallel efficiency with the increasing number of processors (Nikishkov & Kawka 1998).

Initial mesh of the domain is given as a basic input data. The user only needs to create an element for each boundary segment. After the first stage of remeshing, the number of node and element becomes larger depending on error estimation at each initial element. Mesh refining process will continue to work until the permissible optimum mesh arrangement error obtained. In the area with higher stress concentration, the size of element is smaller than areas with low stress concentration.

Calculation result from the first stage is updated and stored in a file for other calculation. Each step of crack propagation, a new boundary segment is formed. The process will continue until all the calculation for elemental stiffness matrix in each loop is performed by slave processors and global stiffness matrix is formed in master processor. Figure 4 shows a flowchart of parallelization adaptive crack propagation process.

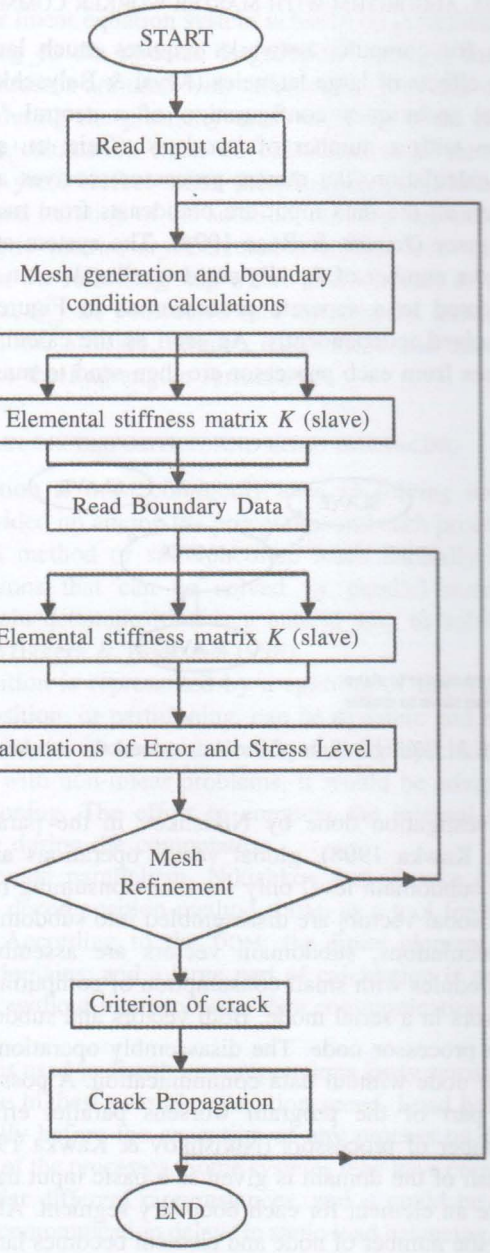


FIGURE 4. Flowcharts for the adaptive crack propagation process using finite element method

PERFORMANCE MEASURES

Amdahl's law predicted very limited improvement in performance because Amdahl claimed that the speed of a multiple-processor computer was limited by its slowest (sequential) part (El-Rewini & Lewis 1998). For a parallel algorithm, in addition to determining the number of computational steps, the estimation of communication overhead is needed.

AMDAHL'S LAW

Assume there will be some parts that are only executed on one processor. The ideal situation would be for all the available processors to operate simultaneously. If the fraction of the computation that cannot be divided into concurrent tasks is f, and no overhead incurs when the computation is divided into concurrent parts, the computation time with n processors is given by $[ft_s + (1-f)t_s/n]$, f is also the inherently sequential fraction of a computation to be solved by n processors (Wilkinson & Allen 1999). Illustrated is the case with a single serial part at the beginning of the computation, but the serial part could be distributed through the computation. Hence, the speed-up factor is given by:

$$S(n) = \frac{t_s}{ft_s + (1-f)t_s/n} = \frac{n}{1+(n-1)f}, \tag{5}$$

where $S(n)$ is speed-up, f is fraction that cannot divided into concurrent parts, t_s is overall time calculation and n is number of processors.

The improvement is theoretically possible in the best case by ignoring overhead and communication costs (El-Rewini & Lewis 1998). For many applications, as problem size increases, fraction of sequential operations decreases. Therefore Amdahl's law plays less of a limiting factor. This equation is known as Amdahl's law.

PARALLELIZATION FOR COMPUTER NETWORKS

For workstation cluster, the communication time will depend on many factors including network structure and network connection. Parallel execution time t_{para} is composed of two parts, computational time (t_{comp}) and communication time (t_{comm}). T_{comp} is the time to compute the arithmetic operations such as multiplication and addition operations of a sequential algorithm. Analysis of the t_{comp} is performed by assuming that all the processors are the same and operating at the same speed, this t_{comm} will depend upon the size of message.

$$t_{comm} = t_{startup} + mt_{data} + t_{idle}. \tag{6}$$

If the number of iterations b, and the size of the message for communication m, the formula for communication time is as follows,

$$t_{comm} = b(t_{startup} + mt_{data} + t_{idle}), \tag{7}$$

where $t_{startup}$ is the startup time (message latency). $T_{startup}$ is time to send a message with no data. It includes time to pack the message at the source and unpack the message at the destination. The term t_{data} are assumed as constants and measured in bits/? sec, t_{idle} is the time for message latency and time to wait for all the processors to complete the works. The performance of parallel algorithms in distributed memory environment is measured by the speed-up factor, efficiency and effectiveness defined respectively as

$$\text{Total Speed-up: } S_p = \frac{T_1}{T_p}; \text{ Total efficiency: } C_p = \frac{S_p}{p};$$

$$\text{effectiveness: } F_p = \frac{S_p}{pT_p}, \tag{9}$$

where T_1 is the CPU time for the best serial algorithm, T_p is the CPU time for parallel algorithm using n processors, S_p is the total speed-up factor for the parallel computation, and C_p is the total efficiency for the parallel algorithm. The temporal performance is given as follows,

$$L_p = T_p^{-1}, \tag{10}$$

where the unit of L_p is work done per micro second. From equations (9) and (10),

$$F_p = \frac{S_p}{pT_p} = \frac{E_p}{T_p} = \frac{E_p S_p}{T_1}, \tag{11}$$

which shows that F_p measure both speed-up and efficiency. Therefore, a parallel algorithm is said to be effective F_p hence,

$$F_p T_1 = S_p E_p. \tag{12}$$

PERFORMANCE RESULTS AND DISCUSSION

In this section, two sets of performance data are presented. The constructed Linux-cluster utilized for the simulation consists of four Pentium 3, 933 MHz CPUs with 512 MB memory. The Linux system has two nodes, four CPUs with each node have two CPUs with shared memory. There are distributed memories between each node with a 100 Mbit 3Com fast Ethernet switch (Li et al. 2002).

Another set for cluster of workstation consists of four Pentium 4 1.6 GHz distributed memory CPUs with 20GB. Despite the method's multiresolution capability, large problem sizes necessitate the use of distributed memory parallel supercomputers to solve the problem (Bao & Bielak 1998). Figures 5 and 6 show the speed-up and efficiency of the model.

The simulation results show that value for speed-up and efficiency for Pentium 3 is higher that Pentium 4. It is because higher CPU speed will decrease the computational time and the ratio of computational to communication times. Architecture for the cluster of workstation also influences the result of calculation. The result also show that cluster of Pentium 3 is better than Pentium 4. For the cluster of Pentium 3, beside connection by network, there is also shared memory in the architecture.

There is no communications cost within processors in shared memory. The communication cost decreased when the value of speed-up is increased. The results present the numerical properties of the parallel solver on the homogeneous architecture of PC with Linux operating systems, connected with Local Area Network (LAN) and using message-passing libraries, MPI

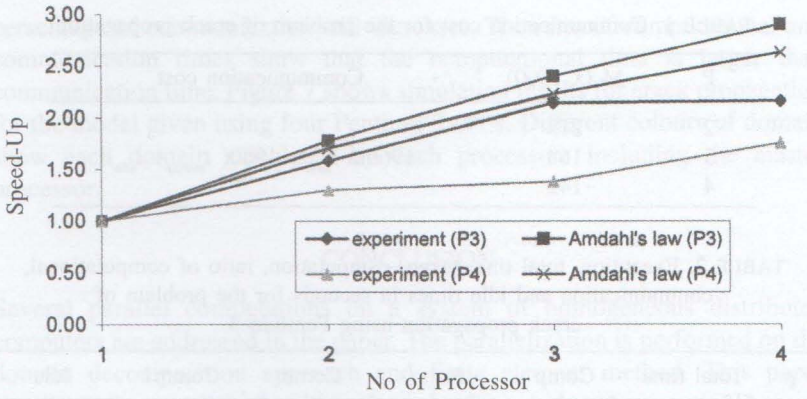


FIGURE 5. Speed-up in the analysis of the model

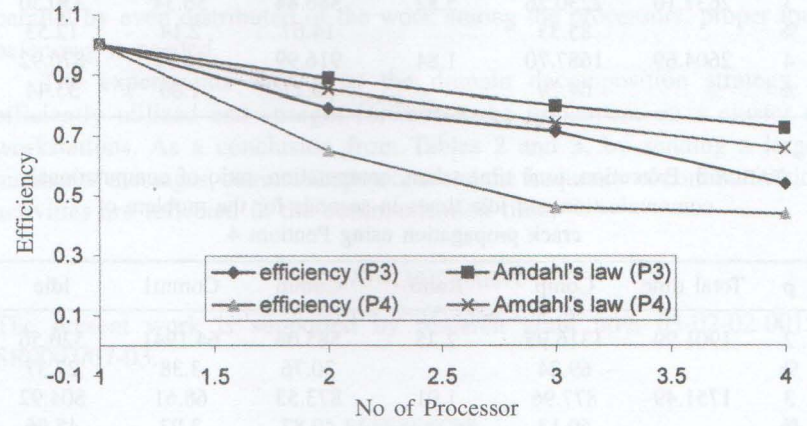


FIGURE 6. Efficiency in the analysis of model

and PVM (Geist et al. 1994). The formula of communication cost for crack propagation solution depends on number of processors and size of data (m) as in Table 1.

From Tables 2 and 3, it is clear that with increasing number of processors, time calculation decreased (Evans & Sahimi 1988). The experiment results show that communication cost depends on number of processors. When the number of processors increases, the communication time increases and the computational time decreases. Both of the experiments show the percentage of idle time is increased. It is because of message latency, load balancing and time to wait for all processors to complete the processes.

In the analysis of crack propagation using Pentium 3, it shows that a gain of communication time is about 13.83 – 14.67 % (2-3 processors). The communication time for Pentium 3 is 35.21% from using from four processors. This is because the idle time is too high when using from four processors until 33.44% compared to using three processors which only 12.53%. When using from four processors for calculation, it include connection using shared memory and distributed memory. These architecture will increase the latency of network and the speed-up become very low.

TABLE 1. Communication cost for the problem of crack propagation

P	M (x 1000)	Communication cost
2	218	
3	164	1960m t_{data} + 1960($t_{startup}$ + t_{idle})
4	142	

TABLE 2. Execution, total time taken, computation, ratio of computational, communication and idle times in seconds for the problem of crack propagation using Pentium 3

p	Total time	Comp	Ratio	Comm	Comm1	Idle
2	3917.35	3375.40	6.23	541.96	39.15	502.81
%		86.17		13.83	1.00	12.84
3	2637.10	2250.26	5.82	386.84	56.34	330.50
%		85.33		14.67	2.14	12.53
4	2604.69	1687.70	1.84	916.99	46.92	870.92
%		64.79		35.21	1.80	33.44

TABLE 3. Execution, total time taken, computation, ratio of computational, communication and idle times in seconds for the problem of crack propagation using Pentium 4

p	Total time	Comp	Ratio	Comm	Comm1	Idle
2	1901.99	1316.93	2.25	585.06	64.1941	520.56
%		69.24		30.76	3.38	27.37
3	1751.49	877.96	1.01	873.53	68.61	804.92
%		50.13		49.87	3.92	45.96
4	1261.92	658.47	1.09	603.45	67.27	536.18
%		52.18		48.22	5.33	42.49

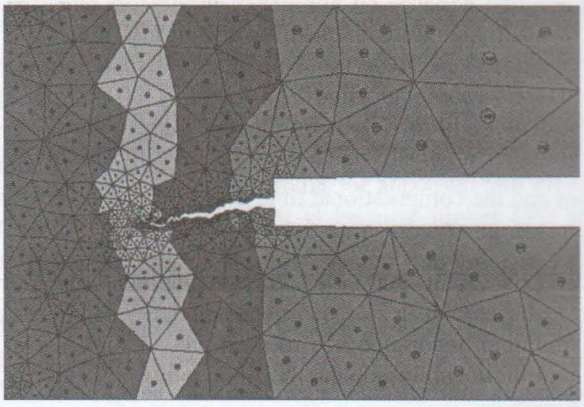


FIGURE 7. Simulation crack propagation using pentium 3

Generally, when the number of processors are increased, data-sending m is decreased but communication between processors is increased. The

persentage of communication still increases. The ratio of computational and communication times show that the computational time is larger than communication time. Figure 7 shows simulation results for crack propagation for the model given using four Pentium 3 CPUs. Different colours of domain show each domain calculated by each processors including the master processor.

CONCLUSION

Several parallel computations on a system of homogeneous distributed computers are addressed in the paper. The parallelization is performed on the domain decomposition approach and finite element method. This paper describes the parallel algorithm for adaptive mesh solution of 2D crack propagation. The algorithms are independent of the underlying boundary value problem and structure. To optimise the computational performance in parallel by even distributed of the work among the processors, proper load balancing is needed.

The experiments show that the domain decomposition strategy is efficiently utilized and straight forward to be implement on a cluster of workstations. As a conclusion from Tables 2 and 3, by sending a larger number of messages, communication cost and the frequency of communication activities are reflected in the communication time.

ACKNOWLEDGEMENT

The present work is supported by research grant IRPA 03-02-02-0015-SR0003/07-03.

REFERENCES

Bao, H. & Bielak, O. 1998. Large-scale simulation of elastic wave propagation in heterogeneous media on parallel computers. *Comp. Meth. Appl. Mech. Eng.* 152: 85-102.

Chandrupatra, T.R. & Belegundhu, A.D. 1997. *Introduction to finite elements in engineering*. Second edition. New Jersey: Prentice-Hall Inc.

El-Rewini, H. & Lewis, T. G. 1998. *Distributed and parallel computing*. New Jersey: Manning Publications Co.

Evans, D. J. & Sahimi, M. S. 1988. The alternating group explicit (age) iterative method for solving parabolic equations i: 2-dimensional problems. *Int. J. Computer Math.* 24:311-341.

Geist, Al., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R. & Sunderam, V. 1994. *PVM: parallel virtual machine, a users' guide and tutorial for networked parallel computing*. Cambridge: MIT Press.

Jaques, M.W.S. & Ross, C.T.F. 1994. Exploiting inherent parallelism in non-linear finite element analysis. *Computer & Structures* 58: 801-807.

Krysl. P. & Belyschko. T. 1997. Object-oriented parallelization of explicit structural dynamics PVM. *Computer & Structures* 66: 259-273.

Li, Y., Sze, S. M. & Chao, T. S. 2002. A practical implementation of parallel dynamic load balancing or adaptive computing in VLSI device simulation. *Engineering with Computers.* 18: 124-137.

Nikishkov, G. P. & Kawka, M. 1998. Porting an industrial sheet metal forming code to a distributed memory parallel computer. *Computer & Structures* 67: 439-449.

Rezende, M. N. D. & Paiva, J. B. D. 1999. A parallel algorithm for stiffness matrix assembling in a shared memory environment. *Computer & Structures* 76: 593-602.

Syifaul, H. 2002. *Permodelan prambatan retak bahan mulur dengan menggunakan analisis unsur terhingga jejaring adaptif.* M.Sc. thesis, Universiti Kebangsaan Malaysia, Malaysia.

Watson, B.C. &d Noor, A.K. 1996. Large-scale contact/impact simulation and sensitivity analysis on distributed-memory computers. *Comp. Meth. Appl. Mech. Eng.* 141: 373-388.

Wilkinson, B. & Allen M. 1999, *Parallel programming: techniques and applications using networked workstations and parallel computers*. New Jersey: Prentice Hall.

Wriggers, P and Boersma, A. 1998. A parallel algebraic multigrid solver for problems in solid mechanics discretisized by finite elements. *Computer & Structures* 69: 129-137.

Department of Mechanical and Materials Engineering
Faculty of Engineering
Universiti Kebangsaan Malaysia
43600 UKM Bangi, Selangor D.E.
Malaysia