Implementation of Parallel Algorithms for Transient Stability Analysis on a Message Passing Multicomputer

C. Hong
Department of Electrical Engineering
Wuhan University of Hydraulic & Electric Engineering
Wuhan, 430072, P.R. China

C.M. Shen
Department of Electrical & Electronic Engineering
The University of Hong Kong
Pokfulam Road, Hong Kong

Abstract: Real time transient stability analysis is a challenging computing problem. In order to speed up the solution of this problem, parallel processing technologies have been applied. In this paper, the implementation of parallel algorithms for transient stability analysis on a message passing multicomputer is described. Both parallelism-in-space and parallelism-in-time are exploited. Test simulations are performed for two large-scale power systems using an IBM SP2 parallel computer. Speedup results are presented to show the performance of the proposed algorithms.

Keywords: Transient stability analysis, parallel algorithm, message passing multicomputer.

I. INTRODUCTION

Time domain transient stability analysis is one of the most computationally intensive power system problems and the industrial need for faster real time solutions to this problem has generated a great interest in finding new algorithms. Due to the availability of a number of commercial multiprocessor machines, a variety of parallel processing algorithms have been investigated [1-9].

The typical serial methods for transient stability simulations use implicit trapezoidal integration to change the differential equations into difference equations which are solved iteratively with the network equations. Traditional methods utilize Newton-like procedures to solve this algebraic problem in a step-by-step mode. Parallelism can be exploited by applying traditional parallel-in-space approaches [5,6]. An alternative to this step-by-step method is to solve transient stability problems on multiple time steps by exploiting parallelism-in-time [2,9]. However, the speedup results reported in [9] suggested that a moderate degree of parallelism-in-time could be used to reduce several times the execution time of transient stability codes. For efficient parallel processing of transient stability problems, it is essential to exploit parallelism inside a single time step and parallelism between the different time steps.

In this paper, parallel algorithms for time domain transient stability analysis are presented. The proposed parallel-in-space algorithm is essentially a parallel version of the Very Dishonest Newton (VDHN) method where a parallel method for solving linear sparse matrix equations has been developed and incorporated into the VDHN algorithm. The proposed network clustering method, which is based on the structure of the factorization path tree associated with the sparse network matrix, can be seen as a generalization of the subtree-to-subcube mapping scheme proposed in [10]. Using the proposed scheduling scheme, coarse-grain parallelism is achieved for effectively solving sparse network equations on message passing multicomputers. In the parallel-in-time approach, a successive over relaxed (SOR) Newton algorithm is implemented in a pipelined-in-time fashion to solve transient stability problems on multiple time steps concurrently. The algorithm take advantage of the traveling window technique [6] to reduce the processor idling deriving from the sequential convergence characteristic of the parallel-in-time algorithm. The method for initialization of state variables over multiple time steps is studied. Parallel algorithms exploiting both parallelism-in-space and in-time have been implemented on an IBM SP2 parallel computer using the Message Passing Interface (MPI). Test results for an actual China power system and a 450-generator, 3021-bus power system are presented and discussed.

II. TIME DOMAIN TRANSIENT STABILITY SIMULATION

The problem of power system transient stability simulation consists, basically, in the solution of an initial value problem for a mixed set of Ordinary Differential Equation (ODE) and algebraic equation:

$$\dot{X} = f(X,V)\,, \tag{1}$$

$$I(X,V) = Y(X)V\,, \tag{2}$$

$$X(t_0) = X_0\,, \tag{3}$$

where X represents the state variable vector and V represents the nodal voltages of the network. In (3), X_0 denotes the state vector at the initial time t_0 in steady state conditions. For time domain solution, (1) can be discretized by the trapezoidal rule and rearranged as:

$$R_G = X_t - \frac{h}{2}f(X_t,V_t) - X_{t-1} - \frac{h}{2}f(X_{t-1},V_{t-1}), \tag{4}$$

and (2) can be written as the following format:

$$R_N = I(X_t,V_t) - Y(X_t)V_t = 0, \tag{5}$$

where $t = 1,2,\cdots,T$ denotes time steps, h represents the step length. Equations (4) and (5) can be compacted in the form:

$$R_t(y_t,y_{t-1}) = \begin{bmatrix} R_G(y_t,y_{t-1}) \\ R_N(y_t) \end{bmatrix} = 0, \tag{6}$$

where $y_t = \left[X_t^T, V_t^T\right]^T$.

Traditional methods solve this nonlinear system in a step-by-step mode using Newton-type procedures. The linearized form of the nonlinear difference-algebraic equations for iterations at each time step can be described as:

$$\begin{bmatrix} R_G \\ R_N \end{bmatrix} = -\begin{bmatrix} A_G & B_G \\ C_G & J_N \end{bmatrix} \begin{bmatrix} \Delta X \\ \Delta V \end{bmatrix}, \quad (7)$$

where $A_G = \frac{\partial R_G}{\partial X}$, $J_N = Y + Y_L = \frac{\partial R_N}{\partial V}$, $B_G = \frac{\partial R_G}{\partial V}$, $C_G = \frac{\partial R_N}{\partial X}$. Y is the network admittance matrix and Y_L is a diagonal matrix obtained from the derivatives of nonlinear load currents with respect to nodal voltages. The R_G and R_N represent the residuals in satisfying (4) and (5) with the current estimates of X and V.

The solution of (7) exploits the block structure of the Jacobian matrix [12]. Reduction of matrix gives the equation for ΔV as:

$$-J_N' \Delta V = R_N' \quad (8)$$

where $J_N' = Y + Y_L + Y_G$, and $Y_G = -C_G A_G^{-1} B_G$ is the sum of the effects of the generators on the admittance matrix. The right-hand side of (8) is the network residuals modified by the effect of doing a Newton-type iteration. That is:

$$R_N' = R_N - C_G A_G^{-1} R_G \ . \quad (9)$$

Once the ΔV are calculated from (9), the ΔX can be calculated by backward block substitution into:

$$A_G \Delta X = R_G - B_G \Delta V \quad (10)$$

for all generators.

Equations (8) and (10) can be solved iteratively to update V and X at each time step. The standard Newton's method is to calculate the Jacobian matrix for every iteration. This method is characterized by quadratic convergence. Many production grade programs iteratively solve these equations using VDHN method which does not update the Jacobian unless the system undergoes significant change or the iteration number exceeds a pre-determined threshold value. In this case, the convergence is no longer quadratic but the solution speed is improved.

III. PARALLEL-IN-TIME FORMULATION

To present the parallel-in-time formulation of Newton-based method for transient stability analysis, (6) is formulated for the solution of T time steps simultaneously:

$$\hat{R}(\hat{y}) = 0 \quad (11)$$

where $\hat{y} = \left[y_1^T, y_2^T, \cdots, y_t^T, \cdots, y_T^T\right]^T$, and $\hat{R} = \left[R_1^T, R_2^T, \cdots, R_t^T, \cdots, R_T^T\right]^T$.

Equation (11) merely represents the nonlinear equations over the T time steps. The rigorous Newton procedure for solution of this nonlinear problem can be formulated as:

$$\hat{y}^k = \hat{y}^{k-1} - \left[\frac{\partial \hat{R}}{\partial \hat{y}}\right]^{-1}_{\hat{y}^{k-1}} \hat{R}(\hat{y}^{k-1}), \quad (12)$$

where k is an iteration index and $\left[\partial \hat{R}/\partial \hat{y}\right]_{\hat{y}^k}$ denotes the global Jacobian of $\hat{R}$ evaluated at $\hat{y} = \hat{y}^k$. The global Jacobian of $\hat{R}$ has a block-lower triangular structure where the bandwidth is equal to *1* since the trapezoidal rule is used. It is apparent that parallel-in-time approaches using this formulation may suffer the waiting problem due to the coupling effects between neighboring time steps. To eliminate the coupling effects, relaxation schemes were proposed to decompose equation (11) in T independent nonlinear systems, each corresponding to a window time step [3,4]. For each of the T independent nonlinear systems, Newton-like procedure were used to solve the algebraic equations. The coupling effects denoted by the off-diagonal blocks of the global Jacobian matrix were concurrently neglected. According to the Gauss-Seidel-Newton method [4], the following formula can be used to update the variables corresponding to each time step of a window:

$$y_t^k = y_t^{k-1} - \left(\frac{\partial R_t}{\partial y_t}\right)^{-1}_{\substack{y_t = y_t^{k-1} \\ y_{t-1} = y_{t-1}^k}} R_t(y_t^{k-1}, y_{t-1}^k), \quad (13)$$

where $t = 1,2,\cdots,T$. It is known from (4) and (5) that the effect of the time step (t-1) on time step t is fully represented in evaluating the residual $R_t(y_t, y_{t-1})$.

The global nonlinear systems can be solved in parallel using a pipelined-in-time fashion [4]. In the present implementation, a task scheduling scheme is presented for implementing the parallel-in-time algorithm (13). The processing arrangement is shown in Fig. 1. After computing y_1^1 on processor P_1, the calculation of the first iteration value y_2^1 of the second time step starts on processor P_2 in parallel with the calculation of the second iteration value y_1^2 on processor P_1. In this computation arrangement, each processor is devoted to the computation of the variables of a single time step for all the iterations required to achieve the convergence. The parallelism is exploited by calculating different iterations of multiply time steps concurrently on parallel processors.

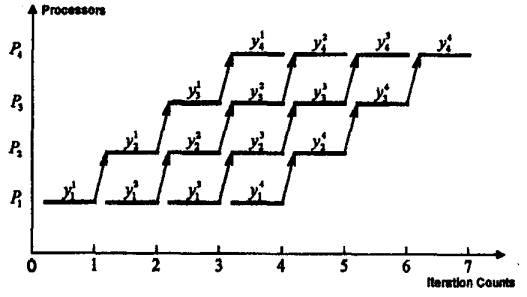

Fig. 1. Sequence of parallel operations for the Gauss-Seidel method

For the first time step of a window, good initial estimates are always available, which are either from the steady state conditions or from the converged values of the transient problem at the time step prior to the first time step of the current window. However, the estimates for the following time steps of the window are much less certain. The proposed computation arrangement introduces an initialization scheme to guess a possible solution of the problem for the state variables at each time step in a window. The y_1^1 calculated on processor P_1 may be used as the initial estimates for the calculation of the second time step

processed on processor P_2. This initialization method can be used to initialize the time steps from 2 to T for each iteration in the current window.

As far as the first time step of a window is concerned, (10) is equivalent to a traditional Newton-like procedure. The convergence of the first time step of a window always precedes the convergence of the subsequent ones. As a result, processors for the earlier steps sit idle or do the redundant calculations after reaching convergence, and thus introduce inefficiencies. To reduce the processor idling, the traveling window technique [6] is used. The converged first time step t of a window will be replaced with the first unassigned time step, which is actually at the end of the new window. Also the proposed initialization scheme can be used to give initial values for the state variables of the new time step.

IV. PARALLEL-IN-SPACE APPROACH

In (7), the coefficient matrix A_G is diagonally blocked since each machine only interacts with the rest of the system through the bus it attaches to. Thus the solution of this portion can be easily parallelized. However, the parallelization of the machine equations only is not enough to achieve high speedup gains because the network solution then becomes a serious bottleneck [5,6].

The coefficient matrix J'_N in (8) is usually extremely sparse, typically incidence symmetric, and with 2×2 blocked structure. The most efficient sequential method for solving this linear sparse matrix equation is accomplished using LU factorization followed by forward and backward substitutions. The factorization path tree is one of the most important tools for understanding the factorization-based solution of sparse linear matrix equations, and it provides appropriate task models for parallel sparse factorization and F/B substitutions [11]. In this approach, we use the factorization path tree to develop a block clustering scheme suitable for parallel implementations on message passing multicomputers. The proposed network clustering method can be seen as a generalization of the subtree-to-subcube mapping scheme [10].

A given factorization path tree can be partitioned into a number of disjoint subtrees by removing a (small) set of nodes from the tree. The removed nodes form a separator of the factorization path tree. Coarse grain parallelism can be achieved by grouping disjoint subtrees so that precedence relationships for each group are wholly contained within the group.

To implement the subtree-to-subcube mapping approach efficiently, a balanced elimination tree is desirable. However, popular fill-reducing ordering methods, such as the minimum degree algorithm and most of its variations, often produce unbalanced factorization path trees. When an unbalanced factorization path tree is partitioned, the sizes of the resulted disjoint subtrees are usually different. Thus, assigning the system buses to processors directly with "one subtree to one processor" scheme may result in much unbalanced computational tasks among processors. To overcome this problem, the disjoint subtrees are grouped to obtain a number of independent clusters which will be assigned to parallel processors with the "one cluster to one processor" scheme. Each cluster may consist of at least one disjoint subtree with the consideration of roughly balancing the computational load among all the clusters.

After the independent clusters have been determined, network buses can be reordered to arrange the network matrix into the bordered block diagonal form (BBDF). The reordering of the optimally ordered buses should be a topological ordering [11] of the factorization path tree where the buses in each cluster corresponding to one or more subtrees are numbered contiguously and the buses in the separator are numbered last. From [11], it is known that this reordering does not alter the precedence relationships in the factorization and F/B substitution phases, thus no extra fill-ins will be introduced and sparsity of the factor matrices will be preserved.

Supposing that the original network has been divided into p clusters using the proposed clustering method, applying the reordering scheme described above, the matrix J'_N will have BBDF structure. Thus, (8) can be formulated as:

$$\begin{bmatrix} J'_{11} & & & J_{1c} \\ & J'_{22} & & J_{2c} \\ & & \ddots & \vdots \\ & & J'_{pp} & J_{pc} \\ J_{c1} & J_{c2} & \cdots & J_{cp} \quad J'_{cc} \end{bmatrix} \begin{bmatrix} \Delta V_1 \\ \Delta V_2 \\ \vdots \\ \Delta V_p \\ \Delta V_c \end{bmatrix} = \begin{bmatrix} R'_{N1} \\ R'_{N2} \\ \vdots \\ R'_{Np} \\ R'_{Nc} \end{bmatrix}, \qquad (14)$$

where subscript c indicates the separator block. It is clear that the p clusters are mutually independent and can be processed in parallel. However, solution of the separator block equations should be processed in serial to avoid massive communication requirements.

V. PARALLEL IMPLEMENTATION USING MPI

A. The Parallel Hardware and Software

The test bed used in this research, the IBM 9076 SP2 system installed in the University of Hong Kong consists of two frames. Each frame contains 16 IBM POWER2 RISC processors with theoretical peak performance 266 MFLOPS per processor. Individual node has its own local 64 MB RAM and local 2 Gbytes of disk storage. Nodes in a frame are connected to each other by a native High Performance Switch (HPS), and the two frames are also linked up by an inter-frame HPS. Also, all 32 nodes are connected by Ethernet. Inter-node communication occurs through explicit message passing.

Each SP2 node runs a full version of AIX, IBM's version of UNIX. It includes all the UNIX features plus specific tools and libraries for programming and executing parallel programs. MPI [17], the Message Passing Interface that is a standard specification for a library of functions implementing the message-passing model of parallel

computation, can be used from FORTRAN 77 and C programs.

In the present applications it is desirable to divide up the available processors to allow different groups of processors to perform computations involved in different time steps while processors in different groups are able to communicate to each other. These features are provided in MPI through communicators that specify communication domains. An intracommunicator can be used for point-to-point communication as well as collective operations within a single group of processors. An intercommunicator can be used for point-to-point communication between two disjoint groups of processors.

B. Implementation of the Parallel-in-Space Approach

In the parallel-in-space approach, the work associated with each time step is partitioned into p subtasks so that p processors can work on one integration step. To keep the communication requirements as low as possible, the machine equations and the network equations of the machine terminal bus are assigned to the same processor. If the terminal bus of a machine is in the separator, the machine equations can be assigned to any one processor with the consideration of balancing computational loads among the processors.

The linear network equation (19) can be solved in parallel using p processors. All the communication between the parallel processors is via the separator block J'_{cc}. In order to make local use of local data as much as possible, a distributed separator block mapping scheme is introduced. The network matrix allocated on the i-th ($i \in [1,p]$) processor is shown in Fig. 2(a). The right-hand vector R'_{Nc} of the separator equations can be scheduled accordingly. The distributed separator block matrix J'_{cc_i} and the network residual vector R'_{Nc_i} satisfy the following requirements:

$$\sum_{i=1}^{p} J'_{cc_i} = J'_{cc} \tag{15}$$

$$\sum_{i=1}^{p} R'_{Nc_i} = R'_{Nc} \tag{16}$$

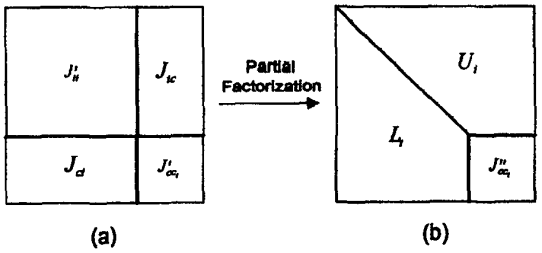

Fig. 2. Partial factorization performed on the i-th processor

During the parallel factorization phase, partial factorization halted before the distributed separator block of the Jacobian matrix can be conducted in parallel, as shown schematically in Fig. 2. Processors $1,\cdots,p$ calculate the L_i, U_i ($i=1,\cdots,p$) factors concurrently. They also calculate the updates of elements belonging to the separator block and transfer the distributed block J'_{cc_i} into J''_{cc_i} ($i=1,\cdots,p$). No communication is needed in this course. Once the i-th processor completes the partial factorization shown in Fig. 2, the partial factors L_i is used to perform forward substitution immediately on the same processor. In this course, the separator residuals vector R'_{Nc_i} will be transferred into R''_{Nc_i} ($i=1,\cdots,p$). After the partial factorization and forward substitution described above have been conducted on the p processors, the following operations must be performed to obtain the full updated separator block matrix and right-hand vector:

$$J''_{cc} = \sum_{i=1}^{p} J''_{cc_i} \tag{17}$$

$$R''_{Nc} = \sum_{i=1}^{p} R''_{Nc_i} \tag{18}$$

The above computations imply that global sum operations are required. An MPI "AllReduce" collective computation operation [17] can be used to accomplish the required communications and computations. MPI global reduction operations can perform a global reduce operation (such as sum, max, etc.) across all the processors of a group. An all-reduce collective computation returns the result of the reduction at all processors in the group. Therefore, after the collective computation operation, each of the p processors obtains a copy of matrix J''_{cc} and vector R''_{Nc} of the separator equations. Then, the solution of the separator equations can be conducted on the p processors concurrently. Though no speedup gain improvement can be achieved when all processors perform the very same calculations, the proposed processing scheme can eliminate the communication requirement during the backward substitution phase since the solution elements of the separator buses are "local" data for all the processors. Therefore, only one MPI collective computation operation is required during each iteration. When the factorized network matrices stored in each processor are reused in a VDHN iteration, there is still only one MPI collective computation operation required during the forward substitution phase, and no communication operation needed during the backward substitution phase.

The proposed scheduling scheme can reduce the communication overhead and increase the amount of the uninterrupted computation during each iteration. The coarse grain scheduling scheme is suitable for implementations on message passing parallel computers.

C. Parallel-in-Space and in-Time Implementation

Fig. 3 illustrate the implementation scheme for the parallel-in-space and in-time approach with p processors working on one integration step and T time steps being

processed simultaneously. The P_t represents the group of p processors working in parallel on the time step t, where $t = 1,2,\cdots,T$. To exploit the parallelism-in-space, the p processors in the group P_t $(t = 1,2,\cdots,T)$ is scheduled according to the scheduling scheme described in the previous subsection. MPI intracommunicator is used for communication among the group of processors where collective computation operations are required. To exploit the parallelism-in-time, MPI intercommunicator is used for communication between processors that processing the same cluster of the network on different time steps. During each iteration, the p processors of group P_t solve the system equations of the current iteration exploiting parallelism-in-space. After the iteration, each of the p processors sends the state variables and voltage vector belonging to the corresponding cluster to the processor calculating the same cluster in group P_{t+1}, and receives data from corresponding processor in group P_{t-1}. To implement the traveling window approach, group P_T should also send messages to group P_1. When group P_1 works on the first time step of the current window, the data received from P_T will not be used. However, when the first time step achieves convergence, the group P_1 will work on the time step $(T+1)$. Then, the data received from processor P_T will be used by group P_1 for solving the new time step. The main feature of the proposed implementation scheme is that the communication pattern is fixed until the last window of the study interval, where the groups of processors at different time steps exit in the order of their convergence and the last step processors exit last.

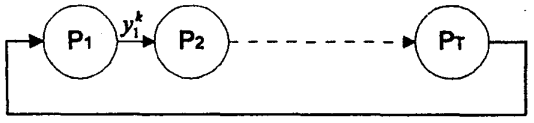

Fig. 3. Structure of the implementation scheme

VI. TEST RESULTS AND DISCUSSIONS

An actual 150-machine, 1007-bus Chinese power system and a 450-machine, 3021-bus power system were tested. Parallel programs were written in FORTRAN 77 plus MPI. A fifth order model was used for the synchronous machines and each generating unit was equipped with an excitation system IEEE DC1 [13]. Nonlinear electrical loads were represented as functions of frequency and voltage. For all the tested cases, a bus three-phase fault with a fault clearing time of 0.06 seconds, a total simulation time of 1 second, and an integration time step of 0.02 seconds were assumed. A convergence tolerance of 10^{-4} *p.u.* on the voltages and on the state variables was enforced. The execution times of the sequential VDHN program on one IBM SP2 node were 3.27 seconds for the 1007-bus system and 10.40 seconds for the 3021-bus system.

When only the parallel-in-space is exploited, the parallel speedup results for the two cases are shown in Tables 1. The speedup $G(p)$ is computed using:

$$G(p) = \frac{T(1)}{T(p)}, \tag{19}$$

where $T(p)$ represents the execution time (wall-clock time) of the parallel algorithm on p processors and $T(1)$ is the execution time of a sequential VDHN algorithm with the same test case and on the same computer.

Table 1. Speedup gains obtained when only the parallelism-in space being exploited for the two test systems

Num. of Processors	2	3	4	6	8
1007-bus system	1.837	2.616	3.175	3.442	3.633
3021-bus system	1.937	2.826	3.649	4.952	5.474

It can be found from Table 1 that the larger system give higher gains, which is certainly a desired feature for parallel computing. Using the proposed parallel-in-space algorithm, the best speedup gain obtained is 5.474 for the 3021-Bus power system when eight IBM SP2 nodes are used. For the 1007-Bus system, the execution time spent to perform parallel transient stability simulation on eight IBM SP2 nodes is only 0.90 seconds for an 1 second simulation interval, which is faster than real time simulation although the speedup gain is only 3.633.

Speedup gains tend to saturate when large parallelism-in-space are exploited. This phenomenon can be analyzed using Amdahl's Law:

$$G(p) = \frac{T(1)}{T_S + T_{OH} + \frac{T_P}{p}}, \tag{20}$$

where T_P and T_S are the shares of parallelizable and the sequential part, and T_{OH} represents the overheads. For the parallel-in-space algorithm, the sequential computing time T_S is mainly derived from the serial processing of the separator network equations and the overhead T_{OH} is mainly the time spent on the communication operations. When more processors are used, the original networks have to be partitioned into more independent, well-balanced clusters by applying the proposed clustering scheme. However, increasing the number of independent clusters may cause a corresponding increase in the size of the separator group, thereby decreasing the parallelizable computational loads, i.e., the sequential overhead T_S will increase while T_P decreases. Moreover, the time T_{OH} generally increases when more processors are involved in the collective operations. From (20), it is quite clear that the increased T_{OH}, T_S and the decreased T_P may cause a fast saturation in speedup gains.

The speedup results for the parallel-in-space and in-time approach with p processors working on one time step and T time steps being solved concurrently were obtained using different values of p and T. The developed programs executed in batch mode, where the application has dedicated use of the allocated nodes. The totally available processors were limited to 28 IBM SP2 nodes since 4 nodes of the parallel computer ran in interactive mode. Table 2 presents some of the test results for the two test systems.

Table 2 Speedup gains of the two test systems with p processors working on one time step and T time steps being solved concurrently

p×T Processors	3×8	4×6	6×4	8×3
1007-bus system	7.249	8.652	9.076	7.920
3021-bus system	7.958	9.969	13.36	11.23

From Table 2, we can see that substantial speedup gains can be achieved via exploiting both parallelism-in-space and parallelism-in-time. The best speedup gain obtained is 13.36 for the 3021-bus system when six IBM SP2 nodes working on one time step and four time steps being processed simultaneously, where the parallel execution time to simulate the transient dynamics for 1.0 second is only 0.78 seconds. This means that on modern parallel computing systems, it is quite possible to achieve the real time transient stability simulation for very large power systems.

With the same number of available processors, exploiting different degree of parallelism-in-space and parallelism-in-time may result in much different speedup gain results, as shown in Table 2. During the tests, we found that the maximum achievable speedup gains of the parallel-in-time algorithm is lower than that obtained with the proposed parallel-in-space implementation of the VDHN algorithm. The main factors limit the speedup gains of a parallel-in-time algorithm are the sequential convergence nature of the algorithm and the load balancing problem when VDHN method is used. Obviously, there is an optimal arrangement for a given case, which is shown from Table 2 to be using 6 nodes working on one time step and 4 time steps being solved concurrently when 24 SP2 nodes are used.

VII. CONCLUSIONS

In this paper, the implementation of parallel transient stability analysis algorithms on a message passing multicomputer were presented. For the proposed parallel-in-space algorithm, the communication overhead and sequential overhead cause a fast saturation in speedup gains when large parallelism-in-space is exploited. High speedup gains are achieved when a moderate degree of both parallelism-in-space and in-time are exploited.

Modern message passing hardware and software provide high computational capability and allow the developed codes portable to other architectures. Effectively utilizing these parallel systems that have several tens of fast processing units, it is quite possible to achieve the real time transient stability analysis for very large power systems.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge Mr. C.M. Woo of the Computer Center at the University of Hong Kong for his helpful suggestions in the programming work.

REFERENCES

[1] IEEE Committee Report, "Parallel Processing in Power Systems Computation," IEEE Trans. on Power Systems, Vol. 7, No. 2, May, 1992, pp. 629-638.

[2] F.L. Alvarado, "Parallel Solution of Transient Problems by trapezoidal Integration," IEEE Trans. on Power Apparatus and Systems, Vol. PAS-98, No. 3, May/June 1979, pp. 1080-1090.

[3] M.L. Scala, M. Brucoli, F. Torelli, and M. Trovato, "A Gauss-Jacobi-Block-Newton Method for Parallel Transient Stability Analysis," IEEE Trans. on Power Systems, Vol. 5, No. 4, November 1990, pp. 1168-1177.

[4] M.L. Scala, R. Sbrizzai and F. Torelli, "A Pipelined-in-Time Parallel Algorithm for Transient Stability Analysis," IEEE Trans. on Power Systems, Vol. 6, No. 2, May 1991, pp. 715-722.

[5] J.S. Chai, N. Zhu, A. Bose and D.J. Tylavsky, "Parallel Newton Type Method for Power System Stability Analysis Using Local and Shared Memory Multiprocessors," IEEE Trans. on Power Systems, Vol. 6, No. 4, November 1991, pp. 1539-1545.

[6] J.S. Chai and A. Bose, "Bottlenecks in Parallel Algorithms for Power System Stability Analysis," IEEE Trans. on Power System, Vol. 8, No. 1, February 1993, pp. 9-15.

[7] G.P. Granelli, M. Montagna, M.L. Scala and F. Torelli, "Relaxation-Newton Methods for Transient Stability Analysis on a Vector/Parallel Computer," IEEE Trans on Power Systems, Vol. 9, No. 2, May 1994, pp. 637-643.

[8] F.Z. Wang, "Parallel-in-time Relaxed Newton Method for Transient Stability Analysis," IEE Proc.-Gener. Transm. Distrib., Vol. 145, No. 2, March 1998, pp. 155-159.

[9] M.L. Scala and A. Bose, "Relaxation/Newton Methods for Concurrent Time Step Solution of Differential-Algebraic Equations in Power System Dynamic Simulations," IEEE Trans. on Circuits and Systems-I: Fundamental Theory and Applications, Vol. 40, No. 5, May 1993, pp. 317-330.

[10] G.A. Geist and E. Ng, "Task scheduling for Parallel Sparse Cholesky Factorization," International Journal of Parallel Programming, Vol. 18, No. 4, 1989, pp. 291-314.

[11] J.W.H. Liu, "The Role of Elimination Trees in Sparse Factorization," SIAM J. MATRIX ANAL. APPL., Vol. 11, No. 1, pp. 134-172, January 1990.

[12] Extended Transient-Midterm Stability Package: Technical Guide for the Stability Program, EPRI EL-2000-CCM-Project 1208, Jan. 1987.

[13] P.M. Anderson and A.A. Fouad, Power System Control and Stability. The Iowa State University Press, 1977.

[14] Marc Snir, et al., MPI: The Complete Reference. MIT Press, 1996.

BIOGRAPHIES

Chao Hong received his B.Sc.(Eng.) and M.Sc.(Eng.) degrees in Electrical Engineering from Wuhan University of Hydraulic & Electric Engineering in 1987 and 1990 respectively. From 1995, he has been a Ph.D. candidate at the University of Hong Kong. He is currently working at Wuhan University of Hydraulic & Electric Engineering as a Lecturer. His research interest is in power system analysis and control, parallel processing in power systems computation.

Dr C.M. Shen received his B.Sc.(Eng.) and M.Sc.(Eng.) degrees in Electrical Engineering from the University of Hong Kong in 1963 and 1965 respectively and his Ph.D. degree in Electrical Engineering from the Queen Mary College, University of London in 1969. Since then he has joined the University of Hong Kong as a lecturer, later retitled associate professor. His research interest is in power system analysis, operation and control. He is an executive committee member of the Specialized Section in Power, IEE Hong Kong and is in the Organizing Committee of the international conferences Advances in Power Systems Control, Operation and Management" (APSCOM) held biennially in Hong Kong since 1991. In APSCOM-95 he was the Chairman of Technical Programs.