PARALLEL PROCESSING FOR SYSTEM SIMULATION

by

HENG-MING TAI, B.S.

A THESIS

IN

ELECTRICAL ENGINEERING

Submitted to the Graduate Faculty
of Texas Tech University in
Partial Fulfillment of
the Requirements for
the degree of

MASTER OF SCIENCE

IN

ELECTRICAL ENGINEERING

Approved

Accepted

May 1982

ACKNOWLEDGEMENTS

I am deeply indebted to Paul Whitfield Horn Professor, Dr. Richard E. Saeks for his direction of this thesis. I would also like to express my sincere thanks to Professors Thomas T. Newman and Kwong Shu Chao for their helpful comments and to Mrs. Pansy Burtis for her fine work in typing this thesis. It is a pleasure acknowledging their help.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

The advent of large scale and very large scale semiconductor integration techniques has provided inexpensive computational capabilities and has generated a great interest in the creation of techniques and algorithms suitable for parallel processing. This enables and also stimulates system engineers and designers to attack problems that once were computationally intractable by implementing systems in which hundreds or even thousands of processors cooperate to solve a single problem. Hence, more and more emphasis is now being placed on parallel computation in the sense that the solution time is reduced while the efficieny with which computational resources are employed is retained.

The developments and applications of the concept of parallelism has been presented in various and disparate fields. Basically, it can be classified into four major levels, namely the bit level, the architecture level, the numerical analysis level, and the system level. On the bit level, for example, we have parallel input/output design and the bit-sliced structure microprocessor. On the architectural level [10], which has proven to be the most fruitful area, cost-effective and highly parallel special-purpose structures for a wide range of problems have been proposed, and many have been built. These machines generally include array processors, associative processors, pipeline machines, data flow processors, multiple CPUs, and the like. To be aware, the data communication problem is the key to the successful exploitation of

parallelism on this level. Studies of parallelism in numerical analysis [13] appeared in the early 1960's, the emphasis in the endeavor has been on the transformation of "classical" sequential nature numerical methods into parallel algorithms. They include algorithms for optimization, root finding, differential equation solutions, and solutions of linear equations. Since problems on system level, such as power system transient analysis [1],[6],[9], the network problem, and the urban planning and traffic problems, are large and complicated in nature, they demand a tremendous amount of computational resources. Hence, further decreasing the computer time required for system simulation is becoming ever more important. Thus, in the thesis, a "highly parallel" system simulation algorithm is presented.

Two simulation schemes which can be directly applied to the component connection model of an interconnected dynamical system are the sparse tableau algorithm and the relaxation algorithm [4]. Roughly speaking, the sparse tableau approach to simulation stacks the various component equations together with the connection model to form a large, highly sparse set of simultaneous equations. With the aid of highly efficient sparse matrix techniques [5],[18] for matrix inversion and nonlinear equation solutions, the sparse tableau algorithm has proven to be extremely effective in a classical sequential computation environment. The second, relaxation, algorithm is built around a predictor-corrector integration scheme and simulates the individual system components one at a time. Thus this algorithm is open to the possibility of solving each component equation independently and concurrently.

In general, a very broad class of physical system to be simulated is described by a set of differential-algebraic equations. Such simulation problems are typically solved by conventional sequential algorithms. In the past decade, however, a great deal of research activity directed at the parallel solution of sets of ordinary differential equations and algebraic equations has been undertaken. Several parallel integration methods for the solution of ordinary differential equation have been proposed, for example, a parallel block implicit method [7], [15],[21], parallel Runge-Kutta and parallel predictor-corrector integration algorithms [13],[14], and a trapezoidal integration scheme [1]. Franklin [7] has also presented a performance comparison of three parallel integration algorithms and developed an appropriate timing formula in a theoretical level comparison. As with the progress of solution of differential equation, various algorithms for the parallel solution of sparse linear equations have been developed in the past few years. Huang and Wing [11],[19] used a triangulated graph and acyclic directed graph forms to obtain a parallel computation model for the solution of linear equations. The ordering technique and the tearing algorithm which decompose a set of equations into block forms are indeed suited for parallel processing [8],[9]. Recent advances in VLSI technology have yielded inexpensive and highly cost-effective processing units and memory chips. Therefore, computer performance will be increased by having many processors operating in parallel on the same problem. As a result, a number of investigations on the utilization of simple, extensible parallel processor structures for efficient solutions have been presented [6],[9],[17],[20].

In this thesis, a relaxation algorithm is proposed for solving large scale system simulation problems in parallel. This algorithm is composed of both a time-step parallel algorithm and a component-wise parallel algorithm and fully exploits the interconnected nature of the system which is characterized by a component connection model. Furthermore, the possibility of this algorithm being implemented with the structures suggested by Pottle [6] and Van Ness [2] is discussed. In chapter 2, the component connection model of a dynamical system and the classical (sequential) relaxation algorithm are reviewed. In chapter 3, we describe not only the time-step parallel algorithm and the component-wise parallel algorithm, but also the combination of these two algorithms to formulate a "highly parallel" system simulation algorithm. Chapter 4 illustrates a number of examples in which the possible trade-offs between efficiency, speed-up ratio, and waiting time are analyzed. Finally, some concluding remarks are made in chapter 5.

CHAPTER II

THE COMPONENT CONNECTION MODEL AND A RELAXATION ALGORITHM

In this chapter the component connection model for a circuit or system is formulated and used as the simulation structure. Although this model has been used before [4],[16], for the sake of completeness we briefly describe it in the first section. The relaxation algorithm upon this model simulated for the predictor-corrector integration method is then presented in the second section.

2.1 The Component Connection Model

The component connection model for an interconnected dynamical system carries a complete description of the system. Basically, this model consists of two separate equations: a decoupled dynamical equation characterizing the components and a coupled algebraic equation characterizing the connections. Furthermore, the component connection model not only unifies the various graphic and diagramatic models, but may be more readily manipulated, both analytically and computationally.

Here the components are described by a set of linear or nonlinear state equations. Let the ith component have input vector a_i, output vector b_i, and state vector x_i. The linear state model for the ith component is the form

$$\begin{aligned} \dot{x}_i &= A_i X_i + B_i a_i \\ b_i &= C_i X_i + D_i a_i \end{aligned} \tag{2.1}$$

Combining the component descriptions defines a composite component state model as

$$\dot{X} = AX + Ba$$
$$b = CX + Da \qquad (2.2)$$

where $a = [a_1,\ldots,a_n]^T$, $b = [b_1,\ldots,b_n]^T$, $X = [X_1,\ldots,X_n]^T$, $A =$ block diag$[A_1,\ldots A_n]$, etc. All vectors are assumed conformable, and they possess values in the appropriately dimensioned Euclidean space. Also, n is the number of components and in general is not equal to the number of states. In the nonlinear case, the state model can be expressed as

$$\dot{X} = f(X,a)$$
$$b = g(X,a) \qquad (2.3)$$

Here $f = [f_1(X_1,a_1),\ldots, f_n(X_n,a_n)]^T$,

$g = [g_1(X_1,a_1),\ldots, g_n(X_n,a_n)]^T$.

The connections are characterized entirely by linear algebraic constraints such as KVL, KCL, and/or other conservation laws. Hence, component and system interconnections take the form

$$\left[\begin{array}{c} a \\ \hline y \end{array}\right] = \left[\begin{array}{c|c} L_{11} & L_{12} \\ \hline L_{21} & L_{22} \end{array}\right]\left[\begin{array}{c} b \\ \hline u \end{array}\right] \qquad (2.4)$$

where L_{ij} are connection matrices; u and y are vectors of system inputs and outputs, respectively. Either equations (2.2) and (2.4) or (2.3) and (2.4) constitute the component connection model

In practice, for "real world" systems the connection matrices are sparse in which sparse matrix techniques can be used to diminish memory storage and computation time. Moreover, the component connection model displays the separate attributes of the system distinctly and independently. Each component is modeled unrelatedly of other components. For

such reasons, therefore, problems of system simulation with the component connection model are well suited for parallel processing. We will investigate the parallel algorithm for system simulation in the next chapter.

2.2 A Relaxation Algorithm

As mentioned in the introduction, the relaxation algorithm simulates a very large system by solving only a single component equation and the appropriate part of the connection equations at a time. Assuming state models for each component, each equation is integrated individually with the results iterated through the connection equations. Hence, simultaneous solutions of the entire set of component connection equations are obtained.

The algorithm for the linear and nonlinear case is more or less equivalent so we only focus on the nonlinear case. The equation under simulation is a set of strictly proper, i.e. g is only a function of X, state equation describing each component of the system

$$\begin{aligned} \dot{X}_i &= f_i(X_i, a_i) \\ b_i &= g_i(X_i) \end{aligned} \qquad i = 1,2,\ldots,n \tag{2.5}$$

together with the algebraic connection equations

$$\begin{aligned} a &= L_{11}b + L_{12}u \\ y &= L_{21}b + L_{22}u \end{aligned} \tag{2.6}$$

In order to explicitly deal with the individual component equation, it is necessary to decompose the L_{11} matrix into distinct submatrices corresponding to the different b_i's as equation (2.7)

$$a = \sum_{i=1}^{n} L_{11}^{i} b_i + L_{12}u \tag{2.7}$$

Here L_{11}^{i} represents the columns of L_{11} corresponding to the subvector, b_i, of b. Since the second equation of (2.6) is an expression for explicitly calculating y from b and u, it may be neglected in the simulation process. Therefore the goal is to solve equations (2.5) and 2.7) simultaneously. Given the system input u and the appropriate initial conditions we first evaluate the component output vectors at t = 0 as

$$b_i(0) = g_i(X_i(0)) \tag{2.8}$$

and then we calculate the component input vectors at t = 0 as

$$a(0) = \sum_{i=1}^{n} L_{11}^{i} b_i(0) + L_{12}u(0) \tag{2.9}$$

These equations construct the initial values for all of the system variables under consideration.

The next step is to solve the individual component differential equation by numerical integration method [3]. We select a predictor-corrector algorithm as the integration method in that it not only allows us freedom to choose the step size and the degree of the interpolating polynomial at each iteration, but offers high numerical accuracy. To solve equation (2.5), we first use an explicit integration scheme as a predictor to obtain an initial estimate for the component state vectors at time t_1. After computing the first estimate of $X_i(t_1)$, initial approximate values for $b_i(t_1)$ and $a(t_1)$ follow by solving equations (2.10) and (2.11)

$$b_i(t_1) = g_i(x_i(t_1)) \tag{2.10}$$

and

$$a(t_1) = \sum_{i=1}^{n} L_{11}^{i} b_i(t_1) + L_{12}u(t_1) \tag{2.11}$$

Second, the improved estimate of $x_i(t_1)$ occurs by a corrector

implemented through an implicit-like integration scheme. This allows us to produce another set of estimates for $b_i(t_1)$ and $a(t_1)$. The process is repeated until the estimates converge to the desired degree of accuracy. Again the initial conditions and the estimates at $t = t_1$ initiate the same type of procedure to estimate the system variables at $t = t_2$. This provides values for $b_i(t_2)$ and $a(t_2)$, etc. Similarly, the system variables at t_k, $k \geq 2$ will be computed.

In practice any of the standard schemes for adjusting step sizes is usable in the relaxation algorithm. In addition, the algorithm has the freedom to choose various step sizes for different components of the system. Furthermore, it is interesting to point out that each component is treated independently. Hence this allows us to deal with them in parallel.

CHAPTER III

PARALLEL ALGORITHMS

In chapter II the component connection model and a relaxation algorithm for system simulation were outlined. Indeed success in developing the highly parallel algorithms depends heavily on the decoupled nature of the system. Even though most large scale systems are either specified in terms of distinct sub-systems, or are decomposable into such sub-systems, the various sub-systems are decoupled and, therefore, there is no obvious mechanism for allocating processors to sub-systems. Fortunately, the component connection model of the system offers a good structure in which the parallelism is inherent.

Two parallel algorithms are thoroughly investigated in this chapter. We first formulate the multi-step predictor-corrector algorithm [3] in which the solution of an ordinary differential equation is obtained at N time steps simultaneously. Then section 3.2 describes the component-wise algorithm which exploits the interconnected nature of the system with separate processors being used to implement the predictor-corrector scheme on each component or sub-system. In the end, the aforementioned two algorithms being combined to formulate a "highly parallel" system simulation algorithm are presented in section 3.3. Moreover, we also discuss the feasibility of using multiprocessor structures [2],[6],[20] to implement the parallel relaxation algorithm. In this thesis, we assume that there are as many processors available as possible.

3.1 Time-step Parallel Algorithm

Parallel methods for solving ordinary differential equations for initial value problems have been devised. Miranker and Liniger [14] have presented a parallel method in which both parallel predictor-corrector and Runge-Kutta integration formulas have been established. Their approach is that the predictor and corrector equations are each solved once for each of the n equations describing the system. Another good candidate for parallel processor implementation is block implicit algorithm [7],[15],[21]. In block implicit methods, the block is considered as a unit calculation in which a number of steps of solution values are obtained simultaneously. The time-step parallel algorithm presented here is similar to the above methods but more convenient to use in our algorithm for system simulation.

Consider the numerical solution of the differential equation

$$\dot{X} = f(X,t) \tag{3.1}$$

under the initial condition

$$X(t_o) = X_o \tag{3.2}$$

For simplicity we assume a uniform step size $h_i = h$ in the formulation of the algorithm. Under this assumption, the time step is simplified to

$$t_n = t_o + nh, \quad n = 1,2,\ldots,N \tag{3.3}$$

To begin with, the forward Euler formula below is used to predict solutions at N time steps

$$X_n^{(o)} = X_o + nh\, f(X_o,t_o), \quad n = 1,2,\ldots,N \tag{3.4}$$

To correct these values, we derive the corrector equations by using Lagrange interpolation formula at the initial value and the predicted values. An example of the corrector equation for N = 3 is given below[15]:

$$
\begin{aligned}
X_1^{(i+1)} &= X_1^{(0)} + \frac{h}{24}\left(9X_0 + 19X_1^{(i)} - 5X_2^{(i)} + X_3^{(i)}\right) \\
X_2^{(i+1)} &= X_2^{(0)} + \frac{h}{3}\left(X_0 + 4X_1^{(i)} + X_2^{(i)}\right) \qquad (3.5) \\
X_3^{(i+1)} &= X_3^{(0)} + \frac{3h}{8}\left(X_o + 3X_1^{(i)} + 3X_2^{(i)} + X_3^{(i)}\right),\ i = 0,1,\ldots m-1
\end{aligned}
$$

The number m is the correction times. In general, several iterations may be necessary to assure that the corrected values are close to the exact solutions. At first glance the processors have to exchange information several times due to the correction times m per block. Once after the N-step solution values have been done, the last solution X_N of the block will be used as a new initial condition to start the same type of procedure to solve the next N-step block. Indeed, the algorithm's parallelism effectively increases the computation time N times providing, of course, that interprocessor communications is not a major factor.

3.2 Component-wise Parallel Algorithm

The approach presented here fully exploits the interconnected nature of the system which is characterized by a component connection model. As stated in section 2.2, the relaxation algorithm treats each component of the system as an independent unit, thus the components of a system can be simulated in a parallel mode.

The component state equations and the connection equation being simulated was given in (2.5) and (2.7). Given the system input and the appropriate initial conditions the first step is to evaluate the component output and input vectors at t = 0 as (2.8) and (2.9). Instead

of directly executing equation (2.9) or using the algorithm for matrix-vector multiplication [12], we compute

$$q_{ij} = L_{11}^{ij} b_j + L_{12}^{ij} u_j \quad , \quad i,j = 1,2,\ldots,n \tag{3.6}$$

where L_{11}^{ij} and L_{12}^{ij} are the ijth entry of L_{11} and L_{12}, respectively. Since q_{ij}, for $i = 1,2,\ldots,n$, can be evaluated in a processor for the corresponding b_j, equation (3.6) will be executed in parallel. After (3.6) is completed, data transfers among processors are needed in order to obtain the values of a. The next step is then to use the relaxation algorithm for step-by-step integration on equations (2.5) and (2.7). Clearly X_j, b_j, a_j, and q_{ij} can be manipulated in the same processor. Therefore, these n components will be simulated by n processors concurrently, provided that there are n processors on hand. In this method, the amount of interprocessor communications at each step is equal to m+2, where m is the correction times. The component-wise parallel algorithm is summarixed as follows:

$$\begin{array}{l}
X_1(0) \to b_1(0) \to q_{11} \to \cdots \to q_{n1} \quad a_1(0) \to X_1^{(0)}(h) \to b_1^{(0)}(h) \to \cdots \quad a_1^{(0)}(h) \\
\quad\vdots \qquad\qquad\quad \vdots \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \\
X_n(0) \to b_n(0) \to q_{1n} \to \cdots \to q_{nn} \quad a_n(0) \to X_n^{(0)}(h) \to b_n^{(0)}(h) \to \cdots \quad a_n^{(0)}(h)
\end{array}$$

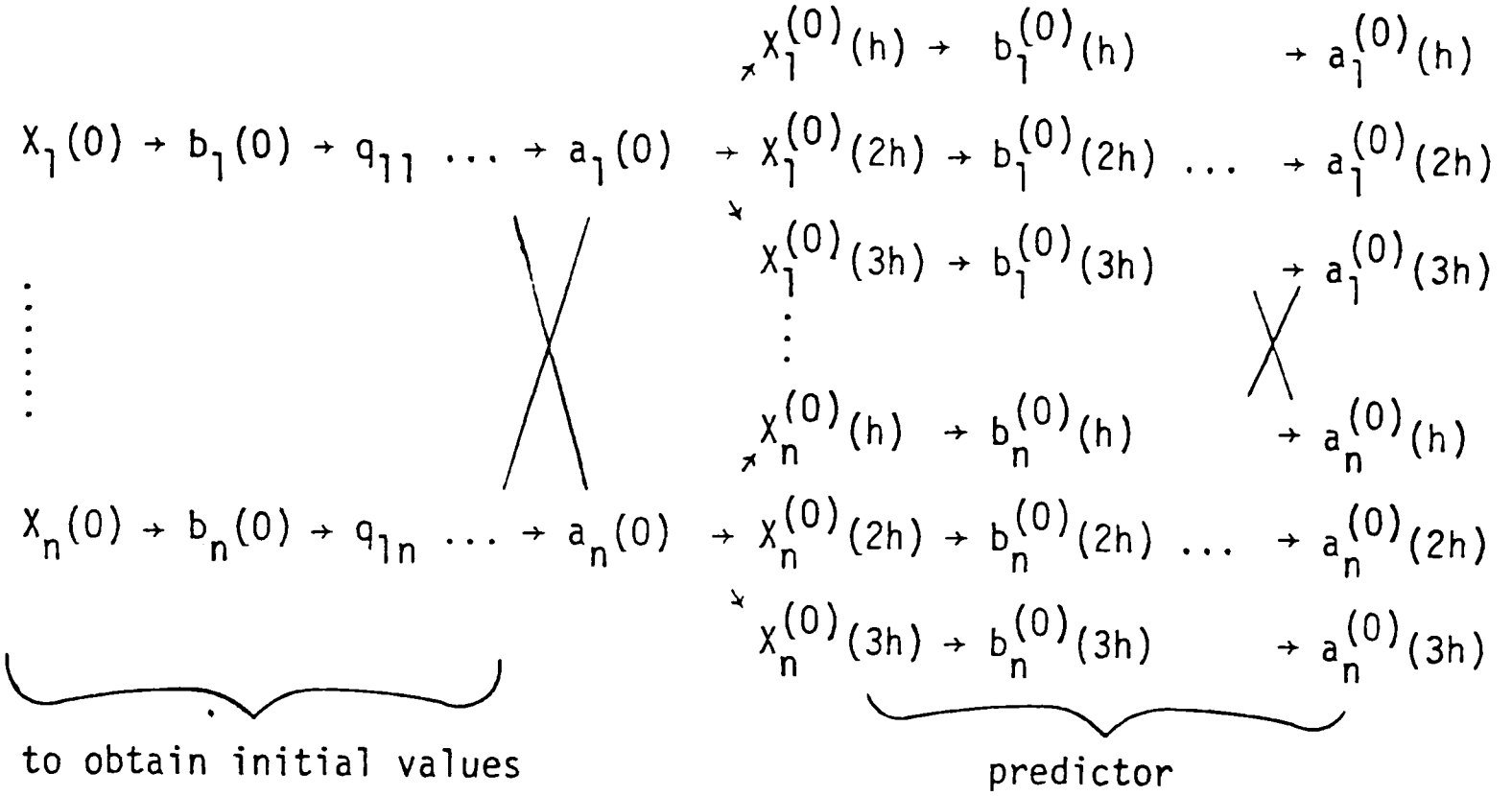
$$\begin{array}{l}
\to X_1^{(1)}(h) \to b_1^{(1)}(h) \to \cdots \to X_1^{(m)}(h) \to b_1^{(m)}(h) \to \cdots \quad a_1^{(m)}(h) \to X_1^{(0)}(2h) \cdots \\
\quad\vdots \\
X_n^{(1)}(h) \to b_n^{(1)}(h) \to \cdots \to X_n^{(m)}(h) \to b_n^{(m)}(h) \to \cdots \quad a_1^{(m)}(h) \to X_n^{(0)}(2h) \cdots
\end{array}$$

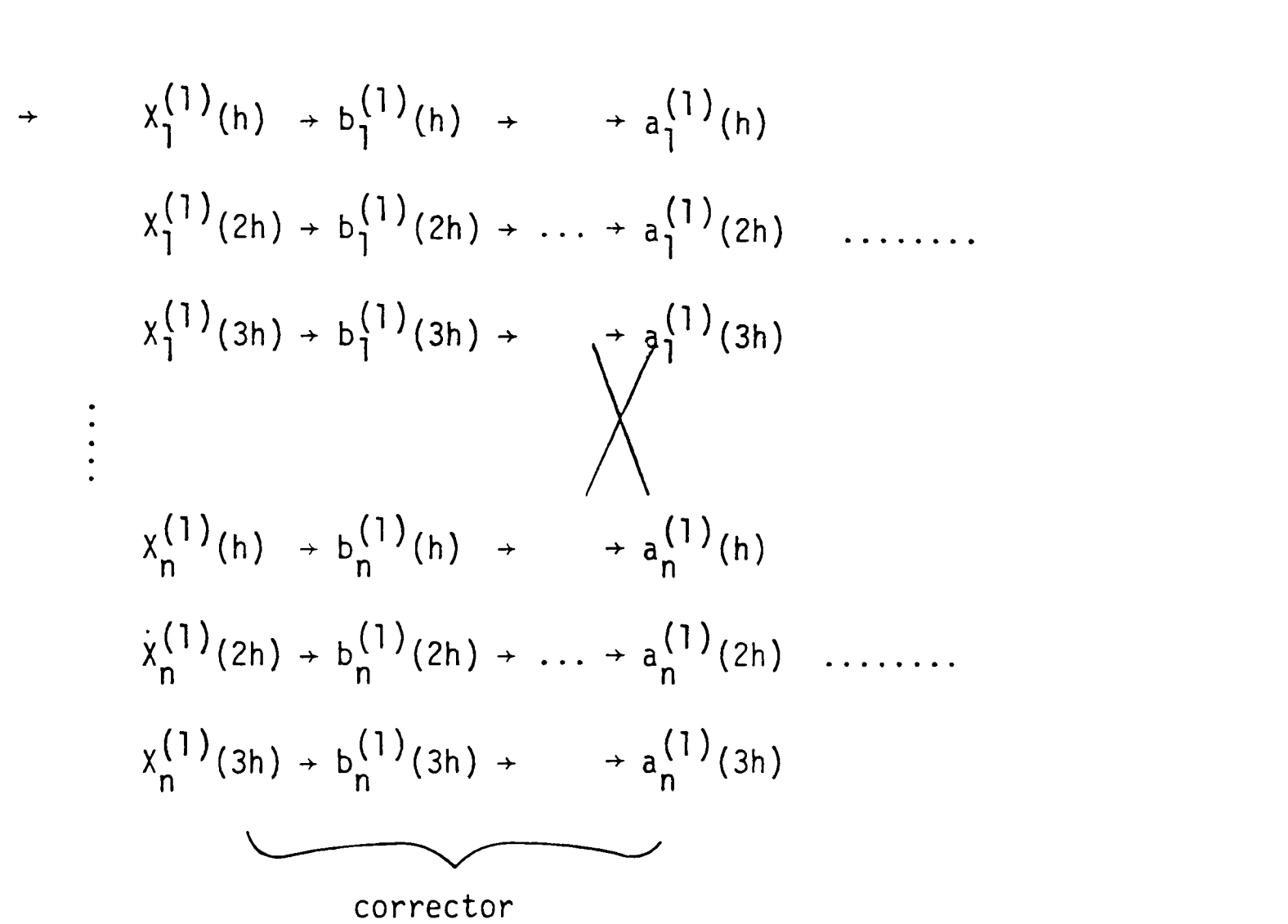
One of the most interesting considerations arising from parallel processing is how effectively and efficiently the algorithm can be executed in parallel on a multiprocessor configuration. The hardware structure chosen for implementing this algorithm is the single bus parallel multiprocessor structure proposed by Pottle [6],[20]. In this structure, an array of n processing elements, each with its own memory unit, communicate via a single common data bus. The advantages of this structure are that it is easily extended and is well suited for implementation with ever faster, ever cheaper VLSI chips.

3.3 Highly Parallel Algorithm

It is interesting that if the above proposed algorithms can be combined together, then the system simulation problems will be solved faster than ever as long as there are many processors available. This approach is a much more powerful algorithm in that it preserves parallelism of both parallel algorithms and may operate simultaneously independently of each other. However, it is apparent that with increasing parallelism there is increasing complexity. As a result, the speed advantage will be reduced.

The procedure of combination is straightforward. If each component state equation is regarded as an ordinary differential equation, then the time-step parallel algorithm presented above can be applied to each component of the system, associated with the component-wise parallel algorithm, to formulate the highly parallel algorithm. This algorithm for 3 time steps is indicated in the following:





Several facts are observed. A great deal of data communications occur' not only in obtaining values of a, but also in each individual component. Furthermore, the increase of the number of multistep will need more

correction times for correctors.

Again, the multiprocessor structure chosen for implementing this algorithm is the partitioned ring structure presented in [2]. The computing elements and their associated local memories are divided into several groups. The processors on each group are connected through a common bus so that a message transmitted by any one of the processors will be received immediately by all of the other processors on that bus. These groups of processors on each bus are connected through a ring structure. With this structure the data communication in each component can be carried out in that group.

CHAPTER IV

SIMULATION

In order to gain more insight into the performance of the algorithm presented, several examples have been selected to illustrate this highly parallel algorithm. The possible trade-offs between speed-up ratio, efficiency, and waiting time are also analyzed. Two examples are the Butterworth low pass filters in which every component has the same complexity. The other one is again a low pass filter, but one component is more complicated than any other.

4.1 Simulation

Since the parallel multiprocessor structure mentioned in the previous chapter does not yet exist, performance evaluation of the algorithm requires simulation on a serial computer. The proposed algorithm was then simulated on a VAX 11/780 computer. We assume that with each component each step is handled out by a processor and each processor possesses the same computing power. In addition to obtaining a reasonably accurate estimation of the solution time, the simulator developed in this study can be used to characterize the requirements of individual processors. In this simulation, each processor is brought into the VAX 11/780 one at a time, and its program executed. After finishing, it is then rolled out, and the processor which is furthest back in its execution time is brough in next. The time spent by a processor consists of the program execution time and the data communication time. Here the synchronization time was negligible.

The solution time of various examples were obtained using a time scale based on the VAX 11/780 computer. To investigate the behaviors and advantages for the concurrent execution of the system simulation problems, the performance evaluation parameters chosen are: 1) speed-up ratio, 2) waiting time, and 3) efficiency. The speed-up ratio is defined as the ratio of the serial solution time (TS) to the parallel solution time (TP). In any algorithm for parallel processing, the speed-up ratio should ideally be proportional to the number of processors used (n). In practice it is difficult to keep all processors busy all the time. Thus, when a processor has executed all the ready tasks it must wait for new data from other processors for further execution. Also the processors must wait for the last processor to finish after completing all the tasks assigned to them. Hence, the waiting time (TW) occurs. The efficiency (EFF) represents the ratio of the difference between the total parallel solution time and the waiting to the total parallel solution time, i.e.

$$EFF = \frac{nxTP - TW}{nxTP} \tag{4.1}$$

While getting the values of these parameters, we can examine how good and/or how much of the advantages gained from the parallel processing.

4.2 Examples

Example 1. Consider the fourth order Butterworth low pass filter shown in Figure 4-1(a). The component connection model can be represented as follows:

$$\dot{X} = \begin{bmatrix} \dot{V}_{C_1} \\ \dot{I}_{L_1} \\ \dot{V}_{C_2} \\ \dot{I}_{L_2} \end{bmatrix} = \begin{bmatrix} \frac{1}{C_1} & & & \\ & \frac{1}{L_1} & & \\ & & \frac{1}{C_2} & \\ & & & \frac{1}{L_2} \end{bmatrix} \begin{bmatrix} I_{C_1} \\ V_{L_1} \\ I_{C_2} \\ V_{L_2} \end{bmatrix} \tag{4.2}$$

$$b = g(x) = X \tag{4.3}$$

$$a = \begin{bmatrix} I_{C_1} \\ V_{L_1} \\ I_{C_2} \\ V_{L_2} \end{bmatrix} = \begin{bmatrix} -1 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} V_{C_1} \\ I_{L_1} \\ V_{C_2} \\ I_{L_2} \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} V_i \tag{4.4}$$

Table 4-1 gives the simulation results for this filter. The time unit is microseconds. The total error to be allowed throughout the integration (ET) was specified, namely $ET=10^{-6}$. We simulate this problem by doing the integration up to 12 steps with h = 0.01. From Table 4-1 it is apparent that the efficiency is more than 90%. This effect is due to almost the same

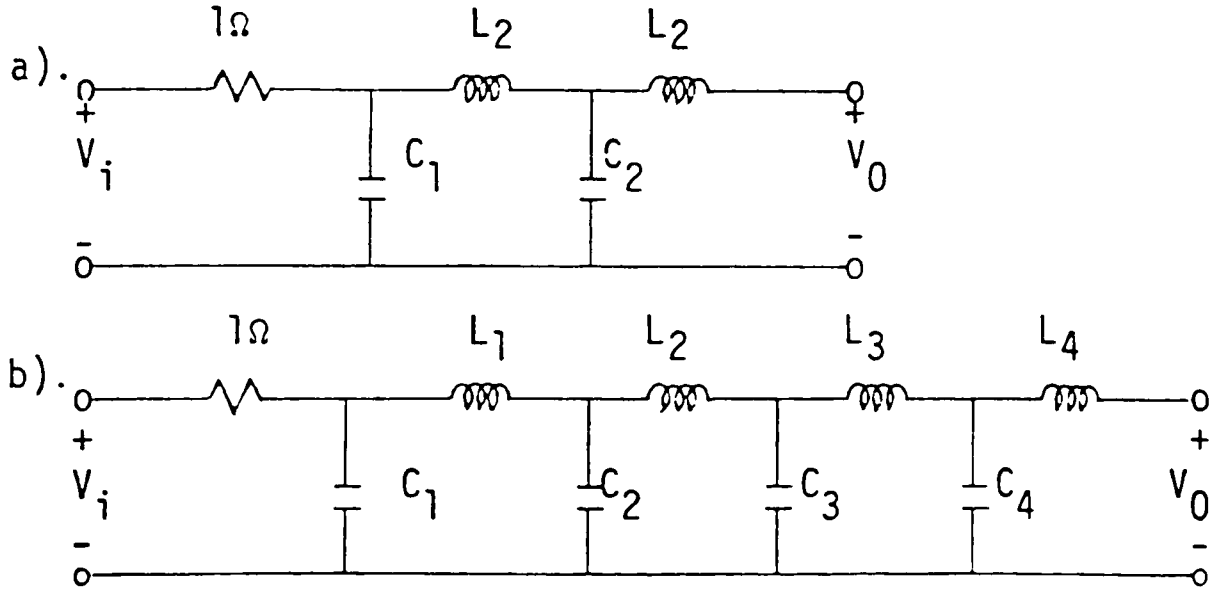


Figure 4-1. Butterworth low pass filters.
(a) Fourth order. (b) Eighth order.

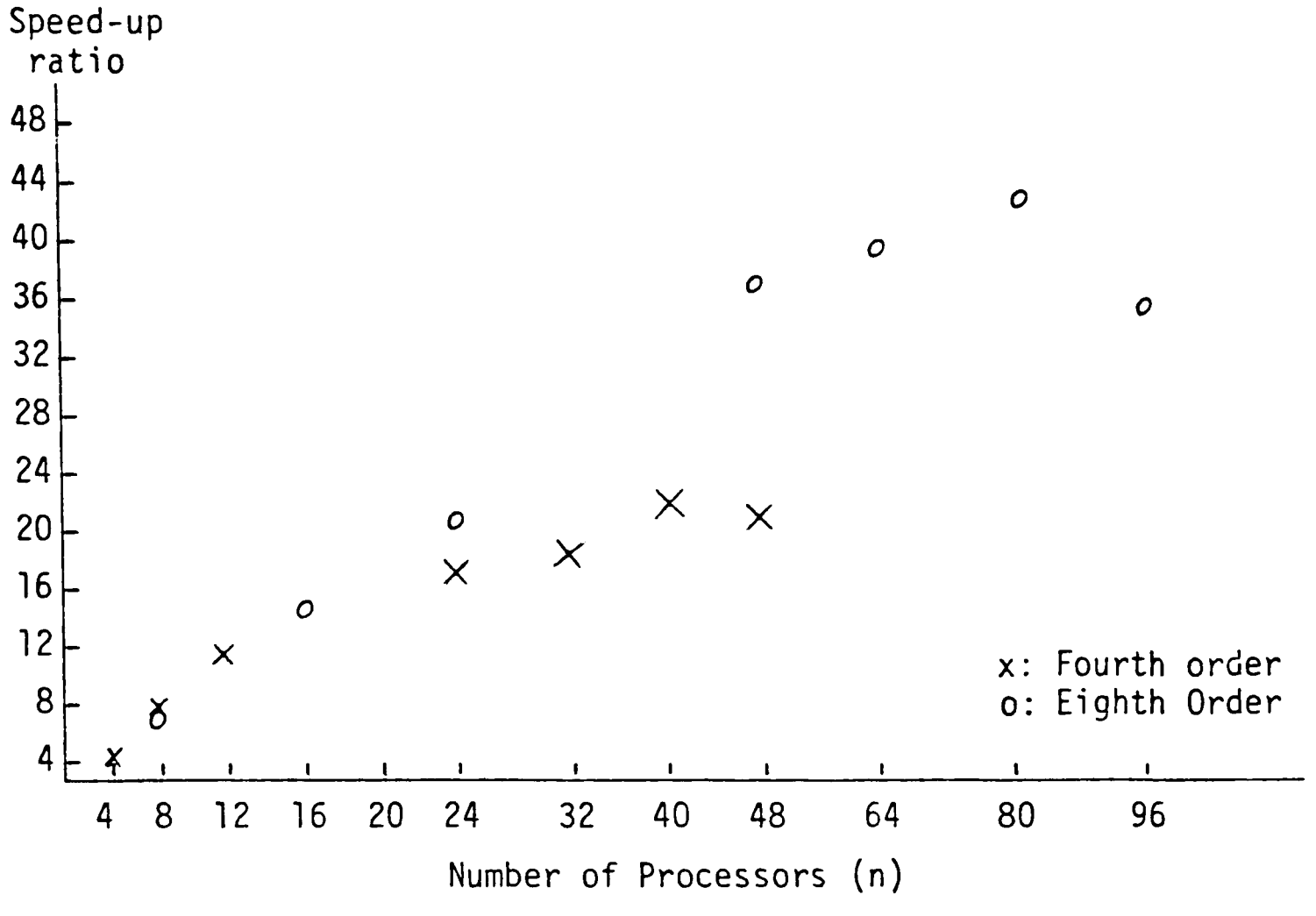


Figure 4-2. Speed-up Ratio vs. Number of Processors for Butterworth Low Pass Filters

complexity of each task to be done. Figure 4-2 shows that the speed-up ratio goes down when the time steps used is greater than 10. It seems that the more steps we apply the more correction times we need so that results are accurate within ET.

Example 2. Consider the Eight order Butterworth low pass filter shown in Figure 4-1(6). This filter has the similar component connection model and the same effect on the performance as example 1. The number of processors used is doubled. The simulation results are given in Table 4-2 and Figure 4-2.

Example 3. Consider a 5-component low pass filter in Figure 4-3 where the complexity of the operational amplifier varies. Table 4-3 and Figure 4-4 gives the simulation results. Note that the existence of the complicated component results in the degradation of the efficiency

Table 4-1. Simulation Results for Fourth Order Butterworth Filter

TS = 2144 μs
Number of Components = 4

Time Unit: microsecond

Number of Processors		TP(μs)	TW(μs)	EFF	Correction times	Speed-up ratio	Efficiency of SUR
1 step	4	552.4	71.2	96.8%	3	3.88	97.03%
2 steps	8	281.6	126.4	94.4%	3	7.61	95.2%
3 steps	12	190.4	170.8	92.5%	3	11.26	93.8%
6 steps	24	119.2	202.8	92.2%	4	17.99	74.9%
8 steps	32	115.5	272.1	92.6%	4	18.56	58%
10 steps	40	95.8	316.2	91.7%	4	22.39	56%
12 steps	48	98	436.8	90.7%	5	21.88	45.6%

Table 4-2. Simulation Results for Eighth Order Butterworth Filter

TS = 4124 μs
Number of Components = 8

Number of Processors		TP(μs)	TW(μs)	EFF	Correction times	Speed-up ratio	Efficiency of SUR
1 step	8	554	200	95.4%	3	7.58	94.76%
2 steps	16	276	253.2	94.3%	3	14.94	93.4%
3 steps	24	187.6	270.8	94.0%	3	21.98	91.6%
6 steps	48	111.6	419.2	92.17%	4	36.95	77%
8 steps	64	103.2	573.3	91.32%	5	39.96	62.4%
10 steps	80	95.8	692.5	90.96%	5	43.05	53.8%
12 steps	96	114.8	1052.8	90.45%	6	35.92	37.4%

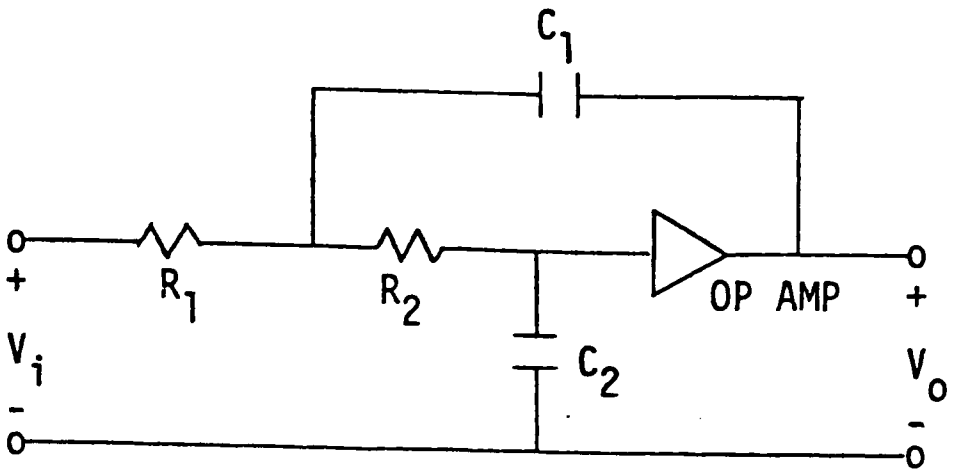


Figure 4-3. A 5-component low pass filter

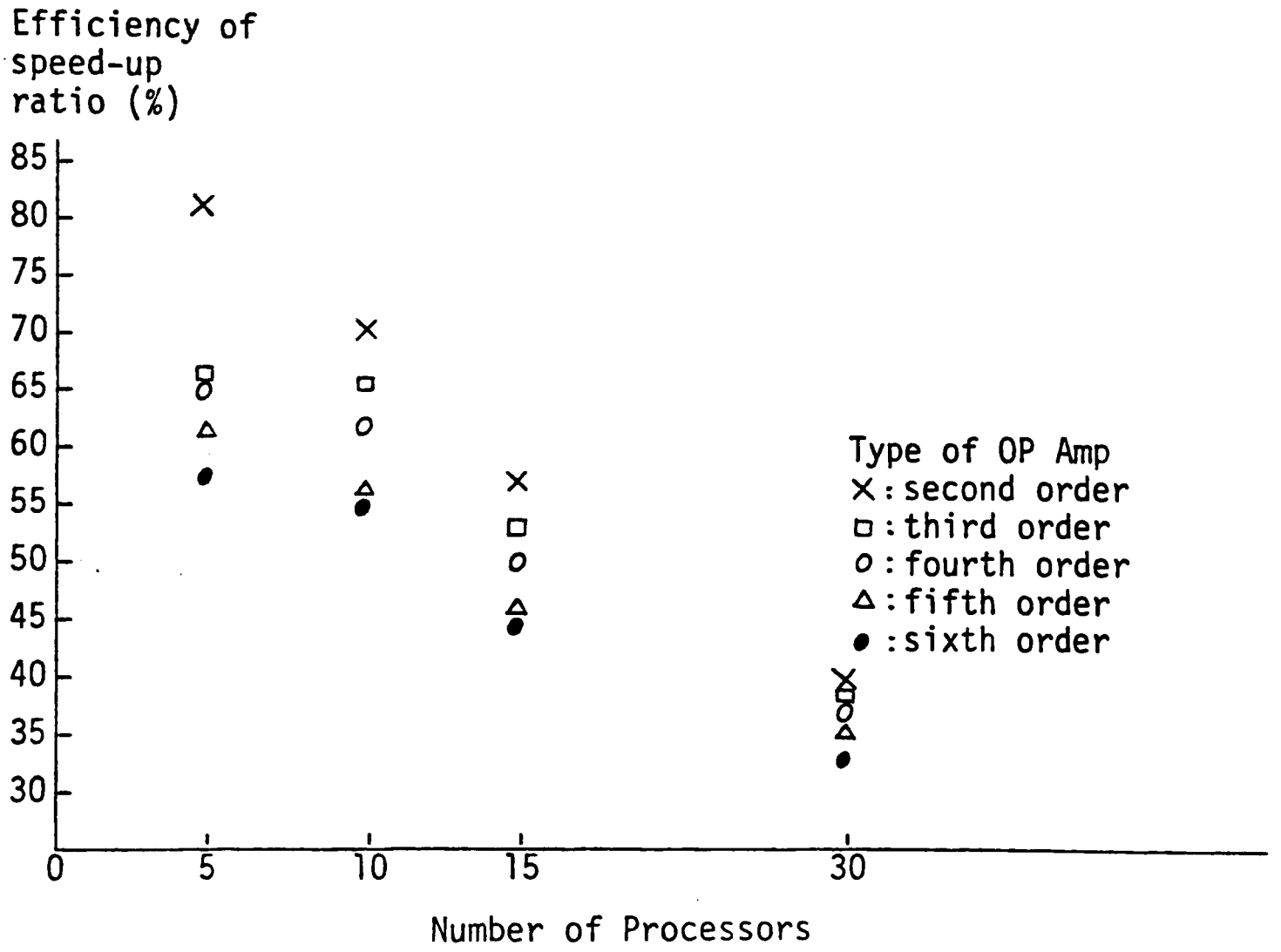


Figure 4-4. Efficiency of Speed-up Ratio vs. Number of Processors for Example 3.

and the speed-up ratio. This is because the solution time of this bottle-processor increases the whole parallel solution time while the serial solution time only increases a little. Also, it causes more and more waiting time. To overcome this drawback, it may be worthwhile to assign more than one processor in this component to reduce the the idle problem.

Table 4-3. Simulation Results for Example 3.

Type of Operational Amplifier	Efficiency				Speed-up Ratio			
	1 step	2 step	3 step	6 step	1 step	2 step	3 step	6 step
Second-order	77.2%	71.2%	70.4%	68.3%	4.08	7.08	8.53	12.24
Third-order	67.2%	66.2%	65.6%	64.4%	3.29	6.51	8.20	11.95
Fourth-order	64.6%	60.9%	60.27%	60.16%	3.26	6.17	7.67	11.52
Fifth-order	60.4%	59.0%	58.1%	57.9%	3.07	5.63	7.16	10.79
Sixth-order	56.3%	55.9%	55.3%	54.6%	2.86	5.60	6.80	10.21

CHAPTER V

CONCLUSION

A highly parallel relaxation algorithm for solving large scale system simulation problems was proposed. This paper has shown that this approach fully exploits the interconnected nature of the system which is characterized by a component connection model. Applications of this algorithm to three examples are studied. The possible trade-offs between efficiency, speed-up ratio, and waiting time are also discussed. Moreover, we have shown the feasibility of implementing this algorithm into a multiprocessor structure to obtain an efficient and cost-effective simulation.

This algorithm has been shown to be a nice parallel simulation under the assumption that all components have the same complexity. However, it is not always the case. The most serious inefficiency in this approach is a result of the waiting time that occurs in the process of simulation. Examples have shown that this decreases the efficiency and speed-up ratio. In other words, advantages of this algorithm are reduced. To improve this algorithm, some other problems are worthy of further research: 1) to reduce the waiting time when solving $\underline{a} = L_{11}\underline{b}$, 2) $\underline{b}$ is a function of $\underline{x}$ and $\underline{a}$ both, i.e., $b = g(\underline{x}, \underline{a})$, 3) $\underline{b} = A\underline{x}$, where A is also sparse, then $\underline{a} = L_{11}\underline{b} = L_{11}A\underline{x}$, is it possible that the matrix multiplication of two sparse matrices can be done in parallel so that performances will be improved?

REFERENCES

1. Alvarado, F.A., "Parallel solution of transient problems by trapezoidal integration," IEEE Trans. on Power Appl. and Syst., Vol. PAS-98, pp. 1080-1090, May/Jun. 1979.

2. Brasch, Jr., F.M., Van Ness, J. E., and S.C. Kang, "Design of multiprocessor structure for simulation of power system dynamics", EPRI Final Report EL-1756, Mar. 1981.

3. Chua, L.O., and P.M. Lin, Computer-aided Analysis of Electronic Circuits: Algorithms and Computation Techniques, Prentice-Hall, Inc. 1975.

4. DeCarlo, R., and R. Saeks, Interconnected Dynamical Systems, New York, Marcel Dekker, 1981.

5. Duff, I.S., "A Survey of sparse matrix research," Proc. IEEE, Vol. 65, pp. 500-535, Apr. 1977.

6. Fong, J., and C. Pottle, "Parallel processing of power system analysis problems via simple microcomputer structures," IEEE Trans. on Power Apparatus and Systems, Vol. PAS-97, pp. 1834-1841, Sept./Oct. 1978.

7. Franklin, M.A., "Parallel solution of ordinary differential equations," IEEE Trans. Comput., Vol. C-27, pp. 413-420, May 1978.

8. Hachtel, G.D., and A. L. Sangiovanni-Vincentelli, "A Survey of third-generation simulation techniques," Proc. IEEE, Vol. 69, pp. 1264-1280, Oct. 1981.

9. Hatcher, W.L., Brasch, Jr., F.M., and J.E. Van Ness, "A feasibility study for the solution of transient stability problems by multiprocessor structures," IEEE Trans. on Power Apparatus and Systems, Vol. PAS-96, pp. 1789-1797, Nov./Dec. 1977.

10. Haynes, L.S., Lau, R.L., Siewiorek, D.P. and D.W. Mizell, "A Survey of highly parallel computing," Computer, Vol. 15, No. 1, pp. 9-24, Jan. 1982.

11. Huang, J.W., and O. Wing. "Optimal parallel triangulation of a sparse matrix," IEEE Trans. Circuits Sys., Vol. CAS-26, pp. 726-732, Sept. 1979.

12. Mead, C., and L. Conway, Introduction to VLSI Systems, Addison-Wesley, 1980.

13. Miranker, W.L., "A Survey of parallelism in numerical analysis," SIAM Review, Vol. 13, pp. 524-547, Oct. 1971.