National Library of Canada
Canadian Theses Service
Ottawa, Canada
K1A 0N4

Bibliothèque nationale du Canada
Services des thèses canadiennes

CANADIAN THESES

THÈSES CANADIENNES

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30.

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

NL-339 (r. 86/06)

Canada

MULTIPROCESSING IN CONTINUOUS SYSTEM SIMULATION

by

Osman Abou-Rabia

A thesis
presented to the University of Ottawa
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
in
Electrical Engineering

OTTAWA, Ontario, 1985

© Osman Abou-Rabia, Ottawa, Canada, 1986.

Permission has been granted to the National Library of Canada to microfilm this thesis and to lend or sell copies of the film.

The author (copyright owner) has reserved other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without his/her written permission.

L'autorisation a été accordée à la Bibliothèque nationale du Canada de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur (titulaire du droit d'auteur) se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation écrite.

ISBN 0-315-33263-8

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

To my mother for all she has given me.

The University of Ottawa requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

ABSTRACT

A special purpose multiprocessor system for large-scale continuous system simulation is described in this thesis. A new variable stepsize parallel integration method is proposed and its performance evaluated. The method provides the means for solving the ordinary differential equations that describe the system being simulated in a parallel manner without the need for partitioning these equations. Stability analysis, complexity considerations and numerical experiments are carried out both with the proposed method and with other members of a related family, and comparisons are made.

A multiprocessor architecture well suited to the proposed method is also presented in which the processors are interconnected through a replicated shared memory. Such an interconnection eliminates both read and write conflicts and allows the data to be instantaneously exchanged among all processors. A simple and effective synchronization mechanism is also discussed.

ACKNOWLEDGEMENTS

The author wishes to express his most sincere appreciation and gratitude to his supervisor, Dr. L.G. Birta for his guidance and encouragement in this research work.

Thanks are due to various members of the secretarial staff of the Department of Computer Science for their help and specially for Carole Ladouceur for typing this thesis.

Financial assistance obtained for this research work from the Departments of Computer Science, Electrical Engineering and the Natural Sciences and Engineering Research Council of Canada is gratefully acknowledged.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter I

INTRODUCTION

1.1 OVERVIEW

Simulation has been described by Shannon [1975] as "the process of designing a computerized model of a system (or process) and conducting experiments with this model for the purpose either of understanding the behaviour of the system or of evaluating various strategies for the operation of the system". Simulation, in general, can be useful whenever it is difficult or impossible to carry out experiments directly on a real system either due to cost, danger, time, or availability of the system.

Simulation, as practiced today, falls into one of three broad categories; namely discrete, continuous or combined. Since combined simulation can be characterized simply as a combination of features from both the discrete and continuous categories, it needs no special consideration. Discrete and continuous simulation have significant differences in several respects. For example, each has its own set of procedures for model conceptualization and each relies on a different set of mathematical formalisms. Furthermore, each has become associated with its own set of software systems (packages) that are tailored to its specific "world view".

Discrete simulation deals primarily with queuing systems in which objects arrive at a service facility, possibly wait in a queue (if all servers are busy), eventually receive service, and finally depart from the facility when the desired service is completed. Manufacturing processes, air traffic control systems and management information systems are typical examples in this category.

Continuous simulation, in contrast, deals with systems that change continuously with respect to time. Since behaviour patterns are generally governed by rates of change, mathematical models for continous systems are based on differential equations. Except for a very limited class of such equations, the only solution method available is numerical integration. Thus, a central work-horse within any continuous system simulation software is the facility that is provided for carrying out this task. Examples of continuous system simulation studies include investigations of aerospace vehicle dynamics, chemical process kinetics and nuclear reactor control. Other applications are described by Adkins and Pooch [1977], and McLean [1980].

The simulation of continuous systems, which is the topic of this thesis, is a computer application area of great importance and is widely used in system analysis, design, and testing, as well as in the training of human operators. In many applications, solution speed is of paramount importance and this aspect is the prime motivation for this research; namely multiprocessing in continuous system simulation.

Initially, simulation studies of continuous systems were carried out, almost exclusively, on analog computers because of the inherent parallelism (hence solution speed) which they provided for the solution of ordinary differential equations (ode's). However the various shortcomings of this technology; e.g. scaling difficulties, limited solution accuracy, relatively high hardware and programming costs and limited storage capability severely restrained the widespread acceptance of this approach. The emergence, in the early 1960's, of the hybrid computer was an attempt to circumvent some of these shortcomings through the blending of analog computer speed with digital storage and processing power. However this approach enjoyed only limited popularity because of the high programming overheads and the inherent idiosyncrasies of the analog computer. As a result, the general purpose digital computers is now the dominant tool in continuous systems simulation studies.

Simulation applications involving continuous system models frequently place severe demands on digital computer performance. Although, the continuing advances in digital computer power has been achieved through advances in circuit-technology and in software, these advances have, by and large, fallen short of the demands arising from the need to handle more extensive and comprehensive mathematical models.

The emergence of large scale integration (LSI), and in particular the development of inexpensive, high performance

microprocessors and memories, have made it feasible to consider an alternate hardware approach for providing computing power for continuous simulation problems; namely, microcomputer operating in parallel [Cyre et al 1977; O'Grady 1980; Brundiers et al 1982; Ercegovac et al 1984]. Two main factors are involved in the evolution of such parallel systems:

1. The need for computers to communicate with one another. This is the most challenging aspect of designing a multiple-computer system since, if communication time is comparable to computation time, the potential of such a system will, to a large extent, be defeated. Parallel approaches for solving the ode's implicit in the continuous system simulation problem can be formulated in a number of ways. These have significant communication requirements. Therefore, considerable care must be taken in designing an architecture to minimize the overhead associated with these requirements.

2. The need for algorithms that can be executed in parallel. Efficient serial algorithms are generally inefficient for multiple processors operating in parallel because they may not ensure that all processors are uniformly active. On the other hand, inefficient serial algorithms may lead to efficient parallel algorithms [Stone 1973].

A common approach for achieving parallelism in the context of digital simulation of continuous systems has been to partition the ode's that describe the system model into groups that are, in turn, solved by individual processors [Korn & Wait 1978; Rideout et al. 1980]. Task partitioning is normally carried out by a compiler. This approach, however, must deal with the difficult problem of evenly distributing the workload over the available processors while at the same time avoiding the creation of significant inter-processor communication requirements.

The alternative of using a true parallel algorithm, which is the basic topic of this thesis, has not been widely emphasized. One of the outstanding features of the proposed approach is the simple way that the communication requirements of the algorithm can be mapped onto a special hardware realization which minimizes the overhead associated with these requirements.

1.2 SOME ILLUSTRATIVE CONTINUOUS SYSTEM MODELS

To provide a clear perspective of the general class of problems that are under consideration in this research project, we briefly outline in this section three representative cases.

(a) Flight Simulation

A classic problem in continuous system simulation has been

the problem of carrying out simulation studies of the flight dynamics of aircraft and spacecraft. Models for this purpose vary widely in their complexity and hence in their domain of usefulness. A reasonably comprehensive six-degree of freedom model is provided in block diagram form in Figure (1.1) which has been adapted from the work of Fogarty and Howe [1968]. A summary of the twelve ordinary differential equations that are embedded in this model are given in Figure (1.2). The various "transformation" and "computation" blocks shown in Figure (1.1) represent wholly algebraic manipulations. Although essential to the model, they are not explicitly listed since our concern is principally with illustrating the nature of the differential equation portion of continuous system models.

(b) Air Cushion Vehicle Control

Amyot [1983] has described an interesting continuous system control problem within the context of the heave performance analysis of an air cushion vehicle (ACV). A schematic representation of the system configuration (adapted from the source paper) is given in Figure (1.3). Figure (1.4) provides a block diagram representation of the various components of the system model that is investigated in the study. The ode portions of the model are contained in the two "dynamics" blocks and these parts of the model are explicitly given below:

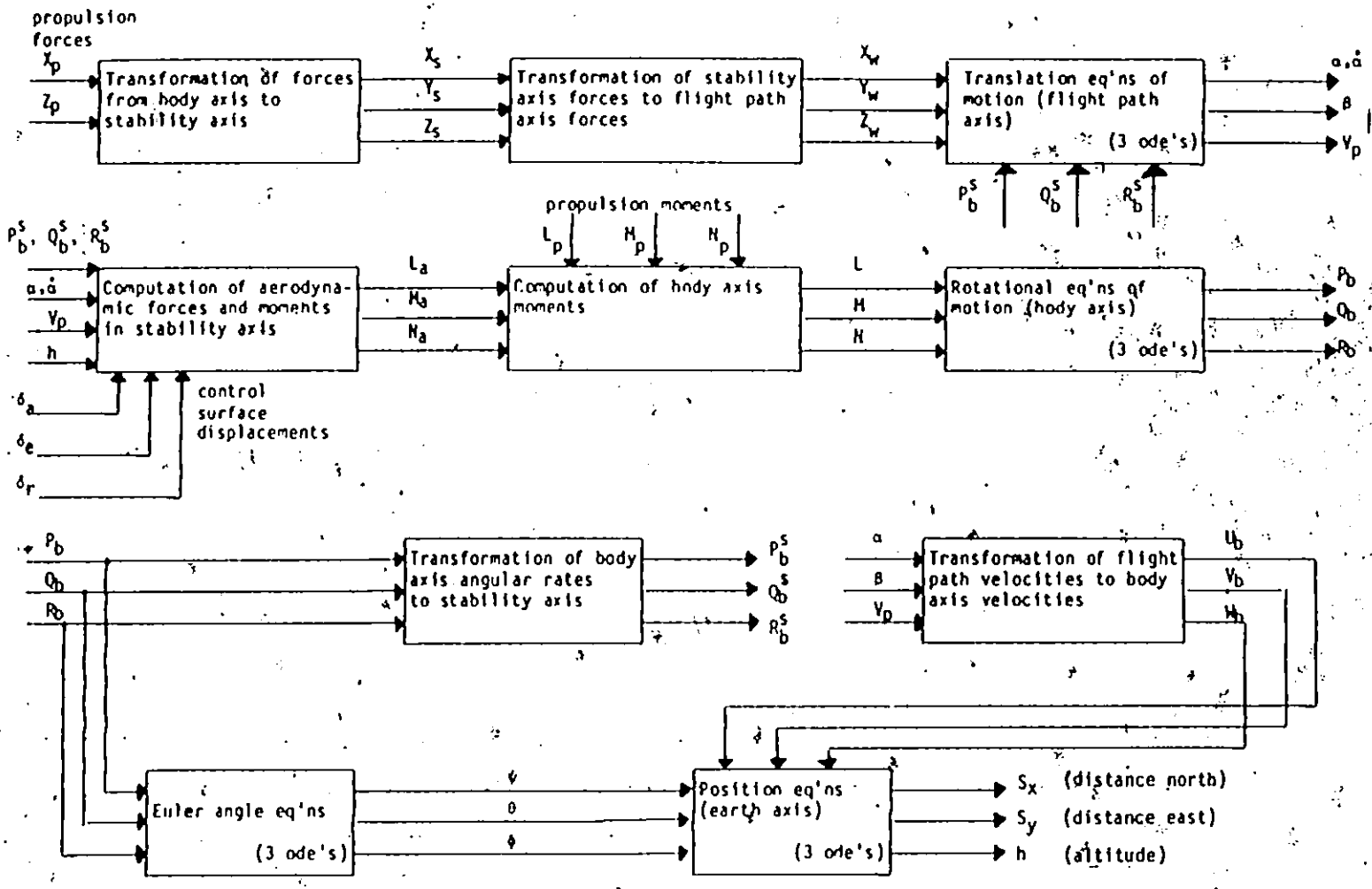

Figure 1.1 Block Diagram of Six-Degree-of-Freedom Flight Simulation

Translational Eq'ns of Motion (flight path axis)

$$\dot{\alpha} \equiv Q_b^s - P_b^s \tan\beta + \frac{Z_w}{mV_p \cos\beta}$$

$$\dot{\beta} = \frac{Y_w}{m V_p} - R_b^s$$

$$\dot{V}_p = \frac{X_w}{m}$$

Rotational Eq'ns of Motion (body axis)

$$\dot{P}_b = \frac{I_{yy}-I_{zz}}{I_{xx}} Q_b R_b + I_{xz}(\dot{R}_b + P_b Q_b) + L$$

$$\dot{Q}_b = \frac{I_{zz}-I_{xx}}{I_{yy}} R_b P_b + I_{xz}(R_b^2 - P_b^2) + M$$

$$\dot{R}_b = \frac{I_{xx}-I_{yy}}{I_{zz}} P_b Q_b + I_{xz}(\dot{P}_b - Q_b R_b) + N$$

Euler Angle Eq'ns

$$\dot{\psi} = (R_b \cos\phi + Q_b \sin\phi)\frac{1}{\cos\theta}$$

$$\dot{\theta} = Q_b \cos\phi - R_b \sin\phi$$

$$\dot{\phi} = P_b + \dot{\psi}\sin\theta$$

Position Eq'ns (earth axis)

$$\dot{S}_x = U_b \cos\theta\cos\psi + V_b(-\cos\phi\sin\psi + \sin\phi\sin\theta\cos\psi) + W_b(\sin\phi\sin\psi + \cos\phi\sin\theta\cos\psi)$$

$$\dot{S}_y = U_b \cos\theta\sin\psi + V_b(\cos\phi\cos\psi + \sin\phi\sin\theta\sin\psi) + W_b(-\sin\phi\cos\psi + \cos\phi\sin\theta\sin\psi)$$

$$\dot{h} = U_b \sin\theta - V_b \sin\phi\cos\theta - W_b \cos\theta\cos\phi$$

Figure 1.2 The ode Portion of the Model Given in Figure 1.1

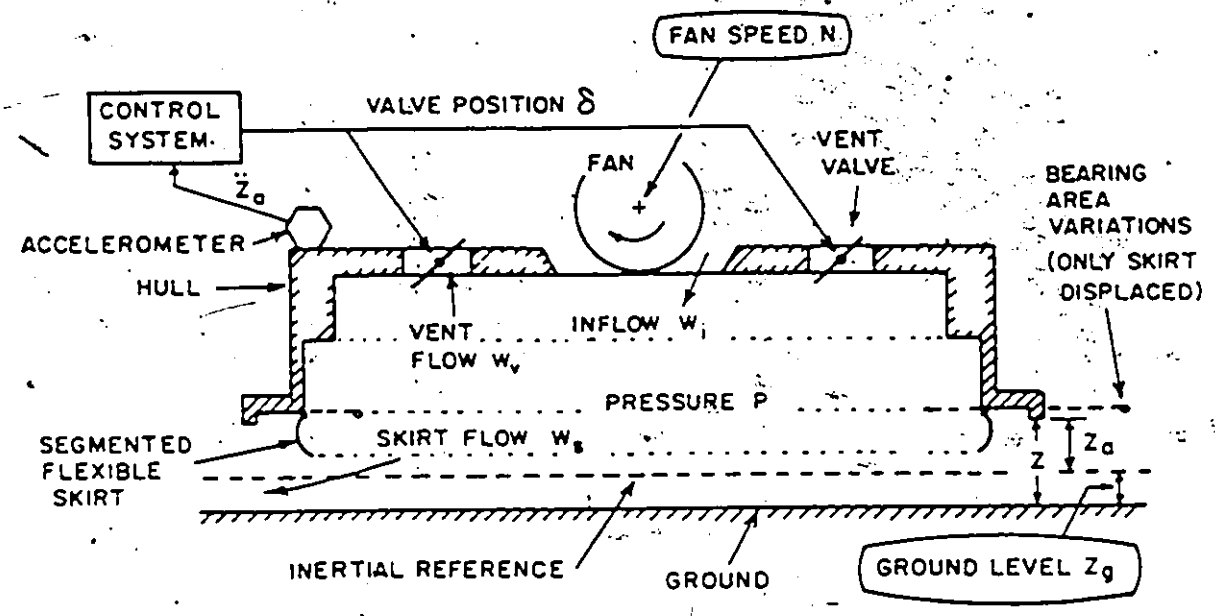

Figure 1.3: Schematic Representation for the ACV Heave Control Problem

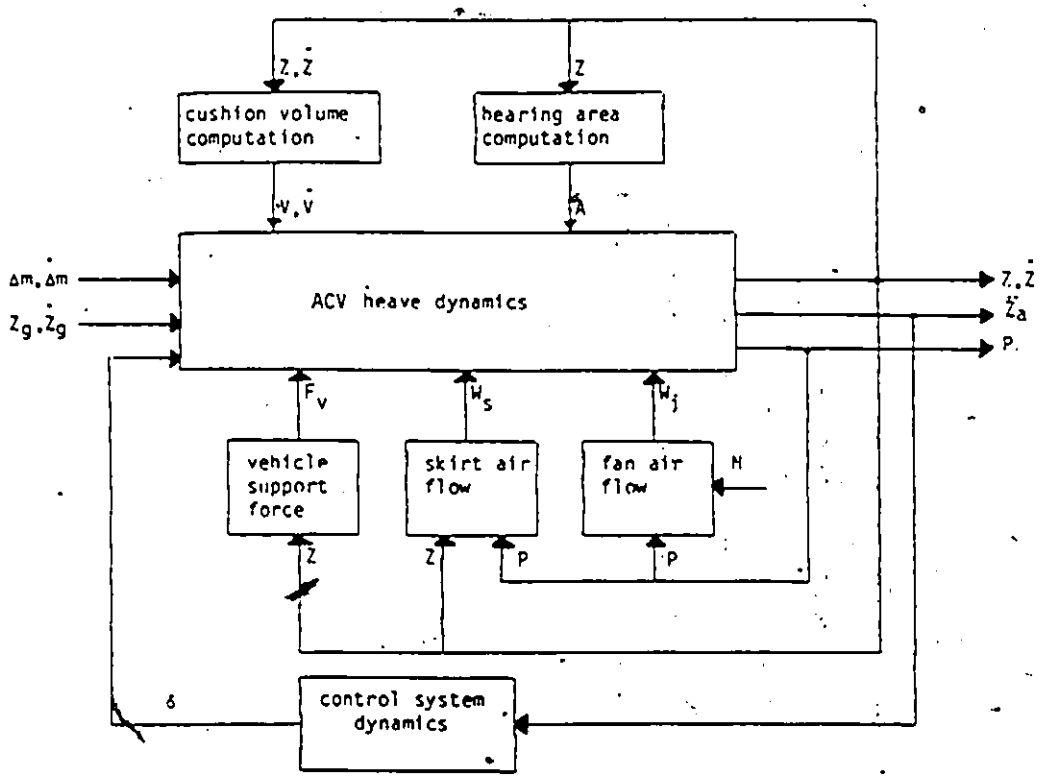

Figure 1.4: Block Diagram for the ACV Heave Control Simulation

ACV Heave Dynamics

$$\ddot{Z}_a = -g - \dot{Z}_a \dot{\Delta m} /(m+\Delta m) + (AP + F_v)/(m+\Delta m)$$

$$\dot{Z} = \dot{Z}_a - \dot{Z}_g$$

$$\dot{P} = C_1(W_i - W_s - C_2 \delta P^{\frac{1}{2}})/V - C_3 \dot{V}(P+C_4)/V$$

Control system dynamics

$$\dot{H} = \ddot{Z}_a - C_5 H$$

$$\dot{\delta} = C_6(f(H) - \delta)$$

In the above equations the C_i's represent constants and m is the vehicle mass. The variable Δm represents an incremental mass that could be added to the vehicle while in motion. H is the output of a high-pass filter within the control system which has $\dot{Z}_a$ as its input. The meaning of the other variables in the above equations is apparent from Figures (1.3) and (1.4).

(c) Heat Transfer Phenomenon

Ode-based continuous system models frequently arise from a discretization procedure applied to an inherently distributed system; i.e., one described by partial differential equations. In this third example, a system of this type is described.

The problem and the system model we outline are taken from

the work of Cheng et al [1977]. Their investigation is concerned with heat transfer phenomenon within a cylindrical copper block having a central core (the problem has its origins in the context of nuclear reactor heat dynamics). A requirement in the study is the determination of the radial temperature profile through the block after the initiation of a cooling fluid flow through the central core. The block is initially assumed to be at a uniform (high) temperature.

When axial temperature variations are ignored, the desired temperature profile, T(r,t), satisfies the one-dimensional Fourier equation in cylindrical co-ordinates; namely

$$\frac{\partial T}{\partial t} = \alpha \left(\frac{\partial^2 T}{\partial r^2} + \frac{1}{r}\frac{\partial T}{\partial r}\right)$$

An alternate (approximate) system model can be obtained by imagining the existence of n concentric rings within the cylindrical block and letting $T_i(t)$ denote the temperature at the i^{th} ring. (This corresponds to a dicretization along the r direction). The following approximations are then introduced.

$$\frac{\partial T}{\partial r} = \frac{T_{i+1}(t) - T_{i-1}(t)}{2\Delta r}$$

$$\frac{\partial^2 T}{\partial r^2} = \frac{T_{i-1}(t) - 2T_i(t) + T_{i+1}(t)}{(\Delta r)^2}$$

where $\Delta r = (r_{out} - r_{in})/n$ (here r_{out} and r_{in} are the outer and inner radii of the solid cylindrical block). With the incorporation of these into the one-dimensional Fourier equation model, the following discretized ode model is obtained:

$$\frac{dT_i}{dt} = \alpha \frac{T_{i-1} - 2T_i + T_{i+1}}{(\Delta r)^2} + \frac{\alpha}{r_i} \left(\frac{T_{i+1} - T_{i-1}}{2\Delta r}\right)$$

$$1 < i < n$$

where $r_i = r_{in} + i\,\Delta r$.

Note that when i=n the corresponding equation in this set contains the variable T_{n+1} which is undefined. This situation is overcome by using the boundary condition:

$$-k(\partial T/\partial r)_{r=r_{out}} = h(T_n - T_{amb})$$

where T_{amb} is the ambient air temperature and h is the heat transfer coefficient at the outer wall. Since

$$\frac{T_n - T_{n-1}}{\Delta r} \approx \left(\frac{\partial T}{\partial r}\right)_{r=r_{out}} \approx \frac{T_{n+1} - T_n}{\Delta r}$$

The boundary condition given above can be written as:

$$-\frac{k}{2}\left(\frac{T_n - T_{n-1}}{\Delta dr} + \frac{T_{n+1} - T_n}{\Delta r}\right) = h\,(T_n - T_{amb})$$

and this equation provides the means for eliminating T_{n+1} from the discretized ode system model.

1.3 OBJECTIVES OF THE THESIS AND MAIN CONTRIBUTIONS

Our primary objective is to develop a special purpose multiprocessor system for continuous system simulation.

Although applicable to all problems in this class, the proposed system is intended to have particular usefulness in both large scale problems (those in which the model involved includes hundreds of equations) and those problems which require time-critical solutions; e.g., flight trainers. In more general terms, the objective is to achieve solutions to continuous system simulation problems in realistic time-frames without recourse to the computing power of mainframe systems.

Two major components are considered for such a system. The first is the development of a parallel integration method which provides for accurate and efficient solution of the ode's that describe the system being simulated. The second is the design of a multiprocessor architecture on which the parallel integration method can be efficiently executed.

Much work has been done in the study of integration methods for a single processor system [Gear 1971; Shampine & Gordon 1975]. However, little attention has been given to the study of integration methods for parallel processing systems. One of the main contribution of this thesis is the development and evaluation of a class of variable stepsize parallel integration methods which

shows good performance over a wide range of accuracy requirements. The attractive feature of this class of methods is that it is intrinsically parallel in nature so that the tasks required by the algorithm can be statically allocated to separate processors and then simultaneously carried out by those processors. The question of optimal task allocation to processors, which occurs in the case of the equation segmentation approach, does not occur here. This simplifies the implementation of the system on which the method is executed and makes task allocation completely problem independent.

Another main contribution is the proposal of a multi-microcomputer system architecture that is well suited to the synchronous nature [Kung 1976] of the proposed class of integration methods. The processors communicate through a special replicated memory organization which eliminates both read and write conflicts and provides for instantaneous data exchange among the processors. A simple but effective hardware synchronization mechanism is also presented.

Other contributions include a stability analysis of various methods of this class as well as an examination of their local truncation errors. Alternate approaches for achieving the variable stepsize behaviour of the methods are also examined. This behaviour has an important impact on the efficiency of integration methods.

1.3 OUTLINE OF THE THESIS

In chapter 2, different approaches for achieving parallelism in the task of continuous system simulation are presented together with a survey of multiprocessor simulation systems.

The development of the class of parallel integration methods that is the main topic of this thesis is provided in chapter 3. This chapter first gives a brief summary of previous work with related parallel methods. A general formulation of the formulas within this class namely; block implicit predictor-corrector (BIPC) methods is provided and two specific alternatives are identified. Their stability characteristics and truncation errors are evaluated. Finally, various means for embedding these underlying formulas within a variable stepsize integration method are formulated.

In chapter 4, various BIPC methods are numerically evaluated over a set of fourteen test problems and over a wide range of solution accuracies. The results are then compared to equivalent results obtained from some conventional serial integration methods using several comparison criteria. One of the BIPC methods, newly proposed in this thesis, shows superior performance.

In chapter 5, a proposal for a hardware implementation of a multi-microcomputer system that is tailored to the class of BIPC methods is presented. An efficient communication mechanism based

on the idea of replicated memories which eliminates both read and write conflicts is incorporated in this architecture. This provides for instantaneous data exchange among the processors. A simple but effective synchronization mechanism is also presented.

Chapter 6 contains the summary, conclusions, and suggestions for future research.

We note finally that some of the contents of this thesis have already been reported or published [Birta & Abou-Rabia 1983; Birta & Abou-Rabia 1983; Abou-Rabia & Birta 1984; Birta & Abou-Rabia 1984; Abou-Rabia & Birta 1985; Birta & Abou-Rabia 1985].

Chapter II

SURVEY OF PAST WORK

2.1 INTRODUCTION

The requirements of engineers and scientists for ever more powerful simulators has been one of the principal driving forces in the development of scientific computers. The attainment of high computing speed as measured by computational throughput has always been one of the most, if not the most, challenging requirements. High-speed becomes of paramount importance in several classes of applications such as aerospace simulation and weather prediction.

In the early days, only analog computers met the demands of high speed and interactive computing within the context of continuous system simulation. Combining both analog and digital computers gave rise to an even more powerful tool: the hybrid computer. However, analog and hybrid computers suffer from problems of scaling, solution accuracy, and programming complexity.

With the appearance of large digital computers and extensive software tools, these problems have been overcome to a large extent. However, despite their capability for high solution accuracy and their other advantages in simulation, digital

computers have a severe shortcoming and inadequate speed due, primarly, to their sequential nature.

In recent years, the availability of low-cost, high performance microprocessors has motivated the idea of interconnecting a number of microprocessors to solve problems of a particular class. Thus, by implementing appropriately designed algorithms on multiple microprocessors, computing times, for many problems, can be expected to be considerably shorter.

Several special purpose multiple processor systems have been proposed for the simulation of continuous systems. However, the potential for success of these systems depends greatly on how the problem is partitioned among the different processors. This matter is discussed further in the following section.

2.2 PARALLELISM IN THE TASK OF CONTINUOUS SYSTEM SIMULATION

The fundamental feature in the description of the model for a continuous system is a set of ordinary differential equations of the form:

$$\dot{z}_1 = f_1 (z_1,\ldots,z_n, t)$$
$$\vdots$$
$$\dot{z}_n = f_n (z_1,\ldots,z_n, t)$$

In this notation, the variables $z_1,\ldots, z_n$ form the state vector and t is the independent variable. The simulation task corresponds to "exercising"; i.e. solving, the model over some

predefined time interval $[t_I, t_N]$ beginning from a given initial state $z(t_I) = z_I$.

Approaches for achieving parallelism in the simulation of continuous systems fall into three different categories; namely, coarse segmentation, fine segmentation [Rzehak 1977] and the use of a true parallel integration algorithm. Coarse segmentation consists of partitioning the system differential equations into segments and then solving each segment independently by a distinct processor. The segments are normally created by a compiler and each segment has an arbitrary number of inputs and outputs. Therefore, the software must be able to create arbitrary segments and distribute them over the available processors. The success of this approach depends, to a great extent, on evenly balancing the workload over the different processors and this is a very difficult and complex task to achieve.

With fine segmentation, the basic arithmetic operations within the model are identified for subsequent allocation to the available processors. Thus, a fairly high degree of parallelism can be expected. Therefore, their is the potential for a large number of simple processors, which require only small private memories. This approach, however, must cope with the inherent problem involved in the interconnection of large numbers of processors as well as with the problem of identifying operations that can be simultaneously executed.

The third approach is based on using a true parallel integration algorithm. In this case, the underlying design of the algorithm assures the existence of tasks that can be simultaneously carried out by separate processors. Partitioning is, in a sense, across the algorithm rather than across the system of equations. Furthermore, the issue of optimal allocation of equations or operations to processors does not occur which makes the implementation of the system simpler and totally problem independent. This alternative, however, has received only minimal attention in the research literature.

2.3 SURVEY OF MULTIPROCESSORS IN CONTINUOUS SYSTEM SIMULATION

Several dedicated multiprocessors for continuous system simulation have been proposed based on the coarse and fine segmentation ideas and these are examined in this section.

Multiprocessor systems based on coarse segmentation have been proposed by Korn [1972] and Korn and Wait [1978]. These systems use a common-bus communication method which is relatively slow and limits the number of processors to 10 or 20. Korn relies on the user to carefully partition the problem in order to minimize interprocessor communications. Such partitioning is, in general, a non-trivial task.

The AD-10 computer [Havranek 1977; Karplus 1977; Gilbert & Howe 1978] is a commercially available simulation computer manufactured by Applied Dynamics International. Several dissimilar

functional processors work in an overlapped and coordinated manner on different aspects of the simulation task such as function evaluation or numerical integration.

WISPAC [Cyre et al. 1977; Rideout et al. 1980] is a parallel array computer developed at the University of Wisconsin. It employs a three-dimensional array of 16-bit microcomputers each connected by a bidirectional serial linkage to its nearest neighbours. Each processor performs the integration associated with a number of state variables. An implementation of the WISPAC system would likely be relatively slow because of the serial communication scheme it uses. Furthermore, there exists a substantial auxiliary software problem of mapping the computational requirements onto the microprocessors.

KCSS [Yoshikawa et al. 1977] is a multiprocessor system, developed at the Keio University in Japan, in which simple, high speed microprocessors execute basic tasks (such as addition, multiplication, and integration of each state) in parallel. A point-to-point parallel interconnection scheme is used. Basically, this system replaces analog computing components (adder, multiplier, integrator) with equivalent digital computing elements on a one-for-one basis. This approach suffers from the same basic shortcoming that is typical of analog computers - namely, the size of the hardware requirement grows with the size of the problem. In addition, the problem of changing the interconnection among the processors, for each new application, needs to be resolved.

Another multiprocessor system designed for the simulation of continuous systems has been constructed at the Swiss Federal Institute of Technology [Halin et al. 1980; Brundiers et al. 1982]. This system is based on the automatic decomposition of the equations within the system model into a large number of low level computational tasks. This is done by a host computer during a compilation step. A Taylor series integration method is used and the necessary higher derivatives of all variables are computed using analytical recursion formulas. Many of these formulas are evaluated by a master computer cooperating with a number of slaves. The starting of processors as masters or slaves, and the bookkeeping for the status of the processors are carried out by a hardware unit called job control unit. The system requires a powerful compiler as well as a sophisticated job control unit.

Another multi-microprocessor simulation system has been implemented at Arizona State University [O'Grady 1980; O'Grady & Wang 1983; 1984]. The system includes NxN processing elements connected through N horizontal buses and N vertical buses. With this arrangement, a data value can be transfered between two processing elements with at most two data movement operations. O'Grady's system requires a complex control mechanism for data transfer and is particularly well suited for matrix calculations.

A parallel computer organization for simulation applications has also been developed at the NASA/Lewis research center [Blech &

Arpasi 1981]. The system consists of up to 10 16-bit processing elements connected through a high speed data transfer bus. One of the processing elements is dedicated to input/output functions while another is used to link low-speed, operator commands with the high-speed simulation activity. Each of the remaining processing elements performs the numerical computations for a predefined part of the simulation task. All processing elements are synchronized; i.e., the data transfer phase takes place when all of them terminate their individual compute cycles. The full potential of such synchronous system is critically dependent on ensuring that the processing elements are occupied for equal periods of time.

A distributed multiple instruction multiple data (DMIMD) processor has been designed and built at the Delft University of Technology for system simulation studies [Decker 1982; Sips 1982; Decker 1983; Andriessen et al 1984]. It consists of a number of processing modules (PM's), each containing a number of fully interconnected processing elements (PE's). Each PE is capable of executing a set of basic parallel operations autonomously using a number of multi-operand processing units. The DMIMD-processor as well as the PM's are loaded from a host computer. The compiler on the host of the DMIMD transforms the program into a task graph indicating the dependencies among the tasks and, consequently, the potential parallelism. The tasks are the basic arithmetic operations such as addition and multiplication. Then the total system definition is subdivided over the available PM's and each PM's host takes care of distributing its task over its PE's and

scheduling their activities. The concept of full interconnectability between the PE's, although it eliminates an important bottleneck in parallel processing, is an expensive alternative. Also, the compiler must be "smart" enough to effectively detect the parallelism within the program.

Data flow architectures have also been proposed for the simulation of continuous systems; e.g. [Karplus & Makoui 1982; Ercegovac & Karplus 1984; Ercegovac et al 1984; Guarini & Cipriano 1984]. The proposal by Ercegovac et al consists of several (4-8) clusters (PE's), each containing one or more microprocessors, memory modules and interface modules. The clusters are connected through a multi-bus communication network. The application programs are represented in a functional language [Backus 1978] to facilitate the extraction of parallelism. The functional programs are partitioned into tasks that are, in turn, allocated to the different PE's. Tasks are activated according to data flow principles [Dennis 1980] where they are ready for execution when their operands arrive. The data flow technique, although it permits a high degree of parallelism, has the fundamental drawback that the concurrency is limited by the communication network which routes results to/from the processing elements.

A special pipelined simulator for continuous systems has been proposed by [McCullough & Linggard 1983] in which a single, high-speed, processing unit is time-multiplexed to compute solutions to sets of ode's.

Parallelism, in all of the above systems, is based on either coarse or fine segmentation, despite the problems associated with each of those approaches. None has been designed around the idea of using a true parallel integration algorithm. This approach, which is the topic of this thesis, has the significant advantage of eliminating the problem of assigning tasks to processors. The approach also lends itself to a simple and straight-forward hardware implementation whose structure is general and equally applicable to all problems within the class.

Chapter III

PARALLEL INTEGRATION METHODS

3.1 INTRODUCTION

The class of continuous system simulation problems that is of interest in this study is characterized by an underlying system model of the form:

$$\frac{dZ}{dt} = \dot{Z}(t) = F(t, U(t), Z(t))$$

where $Z(t)$ is an n-dimensional state vector and $U(t)$ represents an m-dimensional input function ($U(t)$ corresponds to the influence of external forces on the system). In simulation studies, the input function $U(t)$ is always taken to be a known function of time. Consequently, without loss of generality and in the interests of simplifying notation, the effect of $U(t)$ can be assumed to be incorporated within the explicit time dependence that is shown for the derivative function, F.

The fundamental mechanism that is required for carrying out simulation studies in this context, is an effective means for solving the ordinary differential equations (ode's) that are embedded in the model and it is this issue that we explore in this

thesis study. The underlying problem therefore is to generate the solution to a set of ode's having the form:

$$\dot{Z}(t) = F(t,Z(t)) \quad ; \quad Z(t_I) = Z_I \qquad \ldots(\#)$$

from an initial time t_I to a final time $t_N > t_I$, where the state vector Z has dimension n. More specifically, we assume $Z \in \mathbb{R}^n$ and $F: \mathbb{R}^{n+1} \to \mathbb{R}^n$.

The condition which guarantees the existence of a unique solution to this initial value problem is that

$$\left\| F(t, Z_a) - F(t, Z_b) \right\| < L \left\| Z_a - Z_b \right\| \qquad \ldots(3.1)$$

for all $t \in [T_I, T_N]$ and any n-vectors Z_a and Z_b [Henrichi 1962]. (Here $\|\cdot\|$ denotes a vector norm; e.g. euclidean). The requirement (3.1) is known as a Lipschitz condition and the constant L is called a Lipschitz constant. Throughout our considerations, this condition is assumed to be satisfied.

Serial integration methods for the solution of ode's generate one solution value at one point in time by the mean of an integration formula that uses one or more past solution values. This sequential nature makes these methods unsuitable for execution on multiprocessors. Parallel integration methods however, simultaneously generate a set of solution values, at different points in

time, each by a different integration formula. This makes it natural to assign the computation at each time point to a different processor in a multiprocessor environment.

Parallel methods for the solution of ode's have their basis in a class of techniques referred to as "block" methods. The characteristics feature of the methods in this class is that each application of the underlying formulas generates a collection, or block, of k + 1 new equally-spaced solution values; i.e., a k-block. In other words, each basic cycle advances the solution by k new points along the time axis and hence each such block can be considered as a unit calculation.

The underlying process of the block methods can be formulated from the perspective of a single processor (serial implementation) or from that of multiple processors (parallel implementation). Furthermore, such procedures can be formulated either in a "one-step" mode where only the solution at the last point in the block is used to start the next block, or in a multi-step (predictor-corrector) mode in which all the values in the previous block are used to "predict" a solution at each point of the new block. Worland [1976], however, shows that block methods, when applied in the predictor-corrector mode, give higher speed-up than when applied in the one-step mode.

Research with block methods has, for the most part, focused on the one-step methods. The earliest references to the concept

are probably the works of Milne [1953] and Collatz [1966]. In both these cases, the emphasis is within the restricted context of using the approach simply as a means for obtaining a set of starting values for predictor-corrector schemes. The possibility of using one-step block methods, as a solution technique over the entire interval, was first examined in detail by Rosser [1967]. In particular, he presents a set of implicit (i.e., "corrector") formulas for values of k in the range 1 through 8 (included among these is the formulas for k=4 as originally suggested by Milne). The proposed implementation however has a serial (single processor) orientation.

The work of Shampine and Watts [1969] and Watts and Shampine [1972] focuses on convergence and stability properties of one-step methods. A particularly interesting results in [Shampine & Watts 1969] is the demonstration that a particular predictor-corrector formulation of the block methods (with k=2) leads to the same asymtotic behavior as iterating a fourth order one-step method to completion.

Another closely related method (parallel predictor-corrector) has been proposed by Miranker and Liniger [1967], and Miranker [1971]. It has also been studied by Franklin [1978] and by Krosel and Milner [1982]. The algorithm predict and correct current values based on values already evaluated in a previous calculation cycle. Hence, prediction at some points and correction at others can take place simultaneously in contrast to block methods where

the prediction and correction are done at the same points but on different cycles. One of the drawbacks of the Miranker and Liniger algorithm is that it is not well suited for adaptation to variable stepsize scheme; i.e., the capability of varying the integration stepsize with respect to the error which occurs along the solution trajectory. This is due to the spreading of the predictor and corrector equations over two different time steps.

3.2 BLOCK IMPLICIT PREDICTOR CORRECTOR FORMULAS

In this section, the general development of the BIPC formulas is presented. The truncation errors of these formulas as well as their orders are also discussed. Two particular cases for the predictor formulas are identified.

3.2.1 General Development

To avoid undue notational complexity, the development throughout this thesis is carried out in the context of the single first order differential equation

$$\frac{dy}{dt} = \dot{y}(t) = f(t, y(t)) \qquad , t \in [t_I, t_N]$$

$$y(t_I) = y_I \qquad \ldots(*)$$

It should be emphasized that the development can be readily extended to the general case of multiple first order differential equations i.e. equation (#).

An integration method generates approximate values for the solution to (*) at the mesh points $t_1 = t_I + h_1$, $t_i = t_{i-1} + h_i$, $i=2,\ldots,N$ with t_I and t_N given. We use the notation y_i to denote the generated approximation to the true solution $y(t)$ at $t=t_i$. The evaluation of f at (t_i, y_i); i.e., $f(t_i, y_i)$, provides an approximation to $\dot{y}(t_i)$ which we denote by f_i.

As noted earlier, the underlying feature of the block methods is that they provide a set of k equally-spaced solution values during the execution of each cycle of the procedure. These are denoted by:

$$Y = (y_1, y_2, \ldots\ldots y_k)^T$$

Here y_j $(1 \leq j \leq k)$ is the generated solution at $t_j = t_0 + jh$ where t_0 is the right-hand end point of the preceding block and h is the spacing between the solution values within the block. With the BIPC formulation, these k values are produced from y_0 and f_0 (i.e., from the known value at the right-hand end point of the preceeding block) and from a set of tentative ("predicted") solution values denoted by $Y^p = (y_1^p, \ldots\ldots y_k^p)^T$. In particular:

$$Y = \bar{\alpha}\, y_0 + h\, \bar{\beta}\, f_0 + h\, D\, F^p \qquad \ldots(3.2)$$

where $\bar{\alpha}$ and $\bar{\beta}$ are k-vectors, D is a kxk matrix, and F^p is a k-vector whose j^{th} entry is $f_j^p = f(t_0 + jh;\, y_j^p)$. The value of Y^p

is obtained by using y_0 and f_0 and preceding solution information contained in the vector:

$$Y_0 = (y_{-1}, y_{-2}, \ldots\ldots y_{-k})^T$$

where y_{-j} is the generated solution at $t=t_0 - jh$. In particular,

$$Y^p = \alpha\, y_0 + h\, \beta\, f_0 + A\, Y_0 + h\, B\, F_0 \qquad \ldots(3.3)$$

where α and β are k-vectors, A and B are kxk matrices, and F_0 is the k-vector whose j^{th} entry is $f_{-j} = f(t_0 - jh, y_{-j})$.

The k equations represented in (3.3) contain 2k(k+1) free (unspecified) parameters; namely the entries of the arrays α, β, A and B. A set of k(k+2) equations, which partially specify these parameters, can be formulated using the classical method of undetermined coefficients [Carnahan et al 1969]. Specifically, we impose the requirement that (3.3) yields exact results for the cases where the solution to (*) is a polynomial of degree $\leq (k+1)$; i.e., $y(t) = (t-t_0)^r$ for each $r=0,1,2,\ldots(k+1)$. The rational for this process arises from the fact that any function that is continuous on an interval can be approximated to any given precision by a polynomial of sufficiently high degree. For convenience, t_0 is taken to be equal to zero.

Using this procedure it can be shown that the case r=0 (i.e.

$y_i = 1$, $f_i = 0$) yields

$$\alpha = (I-A)\,\bar{u} \qquad \ldots(3.4)$$

where $\bar{u}$ is a k-vector whose entries are all equal to one. When r=1 (i.e. $y_i = t_i$, $f_i = 1$), we obtain

$$\beta = (I+A)\,u - B\,\bar{u} \qquad \ldots(3.5)$$

where u is the k-vector whose j^{th} entry is equal to j. Similarly, when $r > 2$, (i.e. $y_i = t_i^r$, $f_i = r\,t_i^{r-1}$), we obtain the requirement that

$$r\,B\,u^{r-1} = ((-1)^{r-1}\,I + A)\,u^r \qquad \ldots(3.6)$$

where u^r is obtained from u by raising each entry to the power r. Since (3.6) holds for each r in the range 2,3...(k+1), the resulting set of equations can be written in form

$$BU = W\bar{I} + AW \qquad \ldots(3.7)$$

where: $U = [u^1\ u^2 \ldots\ldots\ldots u^k]$

$$W = [w^2\ w^3 \ldots\ldots\ldots w^{k+1}]$$

$$w^r = \frac{1}{r}\,u^r$$

and $\bar{I}$ is the kxk diagonal matrix whose j^{th} diagonal entry is $(-1)^j$. It is shown in Appendix A that U is non-singular and hence (3.7) yields

$$B = (W\bar{I} + AW)\, U^{-1} \qquad \ldots(3.8)$$

Equations (3.4), (3.5) and (3.8) provide k(k+2) linear constraints among the 2k(k+1) parameters in (3.3). Clearly k^2 additional equations are required in order to uniquely specify these parameters and this requirement corresponds to specifications for the entries of the kxk matrix A.

An analogous analysis can be carried out in the context of the corrector equations of (3.2). However, by observing that (3.2) corresponds to (3.3) with A set to zero and $\bar{\alpha}$, $\bar{\beta}$, D playing the roles of α, β and B respectively, the following relationships can be written directly:

$$\begin{aligned} \bar{\alpha} &= \bar{u} \\ \bar{\beta} &= u - D\,\bar{u} \\ D &= WU^{-1} \end{aligned} \qquad \ldots(3.9)$$

(Note also that because the data points relating to the corrector formulas are all located to the right of the base point at t_0, the kxk identity matrix replaces $\bar{I}$).

The general procedure for computing the k solution values within a block is as follows:

1. Compute the entries, y_i^p, within the Y^p vector using (3.3).

2. Compute the entries f_i^p within the F_i^p vector by evaluating the derivative function f in (*) for each y_i^p.

3. Compute the solution values y_i using (3.2).

4. Evaluate f for each y_i to produce f_i.

At the completion of each such step, y_k and f_k become y_0 and f_0, and for each i(i=1,...,k-1), y_i and f_i become y_{i-k} and f_{i-k} of the vectors Y_0 and F_0 respectively in equation (3.3), thereby setting the stage for the next step. It should, however, be noted that the procedure requires a special initialization process because there is no data initially available to form the Y_0 and F_0 vectors.

The important observation in the procedure is that each of y_i^p and y_i together with its derivative, with i=1,2,...k, can be computed independently by a distinct processor provided information exchange takes place at the end of steps 2 and 4. Because, each processor carries out exactly the same computations, but with different data; they become ready to exchange data at the same instants of time. These points are called synchronization points and are shown by the arrows in Figure (3.1) which gives the general timing diagram for a k-block implementation. Any architecture to efficiently support this procedure must therefore be able to handle this simultaneous data exchange among all processors. Note in particular this precludes the use of a single bus common memory

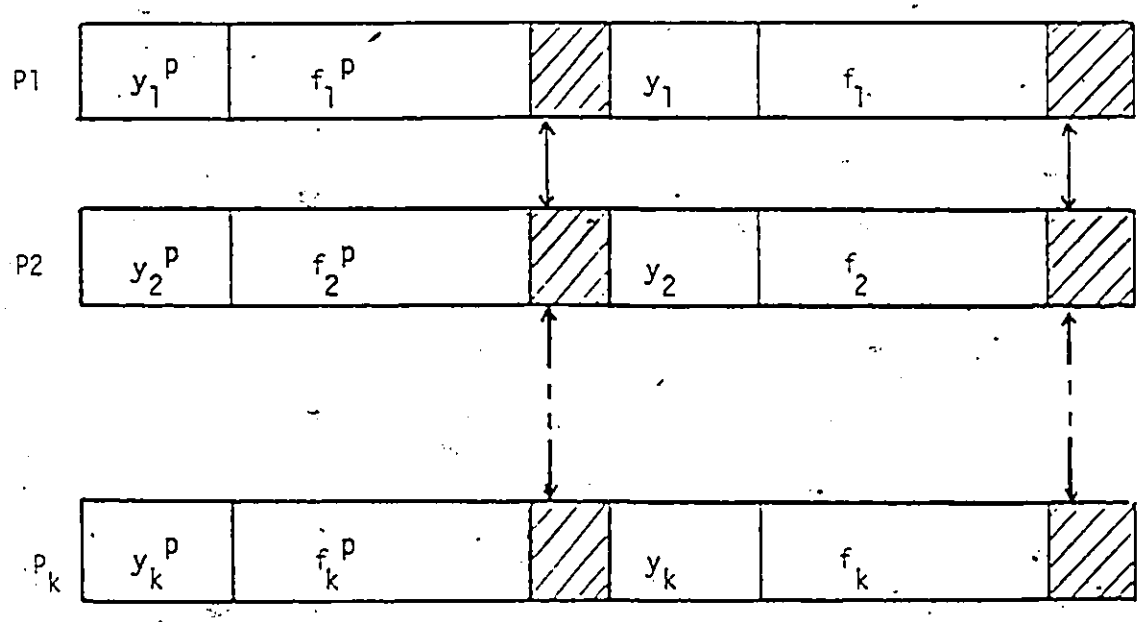

Figure 3.1: General Timing Diagram for a k-block Method

configuration. A detailed description of a multiprocessor system architecture which has this necessary feature for the efficient implementation of the BIPC approach is given in Chapter 5.

Increasing k, the number of points per block, has two important consequences. The first is that the degree of parallelism is increased because each of the k solution values is simultaneously computed by a separate processor (as shown in Figure 3.1). The second effect is that the order of both the predictor and corrector formulas (i.e. equations (3.3) and (3.2) respectively) is increased. The apparent advantage of this second feature is, however, offset to some extent by a reduction in the stability region of the formulas as is discussed in Section 3.3.

3.2.2 Truncation Error

Through the construction procedure, each of the corrector and predictor formulas contained in equations (3.2) and (3.3) has been made exact for a polynomial of degree (k+1) and therefore is said to be of order (k+1). It is important to note here the relation between the block size k and the order of the formulas; i.e., increasing k increases the order of the formulas. When $y(t)$ is not a polynomial or is a polynomial of degree greater than (k+1), then the formulas yield inexact results. The dominant error is related to the first neglected polynomial $y(t) = t^{k+2}$. On this basis, we can write

$$y(t_i) - y_i = T_i(h^{k+2}) + \text{higher order terms} \quad \ldots(3.10)$$

and

$$y(t_i) - y_i^p = T_i^p(h^{k+2}) + \text{higher order terms} \quad \ldots(3.11)$$

where y_i and y_i^p are given by the i^{th} equations of (3.2) and (3.3) respectively. The terms T_i and T_i^p are frequently called the principal local truncation errors. The truncation error is local in the sense that it arises from an application of the formulas on the assumption that no errors have been made in the past data. T_i and T_i^p have the form

$$T_i = \phi_i \, y^{[k+2]}(\zeta_i) \qquad ; \zeta_i \in [t_0, t_i]$$

and

$$T_i^p = \phi_i^p \, y^{[k+2]}(\zeta_i^p) \qquad ; \zeta_i^p \in [t_{-k}, t_i]$$

The term ϕ_i is calculated by letting $y(t) = t^{k+2}$ in equation (3.10), and replacing y_i by its value from equation (3.2). Substituting $y_i = t_i^{k+2}$, $f_i = (k+2)\, t_i^{k+1}$, and $y^{[k+2]} = (k+2)!$ yield, after some manipulation, that

$$\phi_i = C_i \, h^{k+2}$$

where

$$C_i = \frac{(i)^{k+2} - (k+2) \sum_{j=1}^{k} d_{ij} * (j)^{k+1}}{(k+2)!} \quad \ldots(3.12)$$

In a similar manner, it can be demonstrated that

$$\phi_i^p = C_i^p \, h^{k+2}$$

where

$$C_i^p = \frac{(i)^{k+2} - (-1)^{k+2} \sum_{j=1}^{k} a_{ij}^* (j)^{k+2} - (-1)^{k+1} (k+2) \sum_{j=1}^{k} b_{ij}^* (j)^{k+1}}{(k+2)!} \quad \ldots(3.13)$$

Thus, the principal local truncation error of the corrector formulas is given by

$$T_i = C_i \, h^{k+2} \, y^{[k+2]}(\zeta_i) \quad ; \zeta_i \in [t_0, t_i] \quad \ldots(3.14)$$

and that of the predictor formulas by

$$T_i^p = C_i^p \, h^{k+2} \, y^{[k+2]}(\zeta_i^p) \quad ; \zeta_i^p \in [t_{-k}, t_i] \quad \ldots(3.15)$$

Notice, in particular, that the error in both cases is of the same order.

The above results apply for each i=1,...k in the case of the predictor formulas; however, in the case of the corrector formulas, they apply only for i=1, ...,(k-1). An interesting feature of the BIPC methods (at least for the cases where k<10) is that the corrector formulas at the right hand end point (i.e. when i=k) yield a result which is one order higher. The principal local truncation error at this point is in fact

$$T_k = C_k \, h^{k+3} \, y^{[k+3]}(\zeta_k) \quad \zeta_k \in [t_0, t_k]$$

$$\text{where } C_k = \frac{(k)^{k+3} - (k+3)\sum_{j=1}^{k} d_{kj} * (j)^{k+}}{(k+3)!}$$

3.2.3 Specification of the Free Parameters in the Predictor Formulas

The k(k+2) equations of (3.9) provide the means for uniquely determining the parameters in the corrector equations of (3.2). However, in order to complete the specifications for a procedure, additional conditions need to be formulated which will enable values to be established for the k^2 coefficients contained in the A matrix of the predictor equations of (3.3). An obvious general guideline here is to make this selection in a way that enhances the performance of the overall solution process.

3.2.3.1 EWP Form

This formulation is based on choosing the entries of the A matrix to minimize round-off error (or noise). This error arises from the finite precision with which numbers are represented within a computer. Following a suggestion by Hamming [1973], we use the quantity

$$N_i = (\alpha_i^2 + \sum_{j=1}^{k} a_{ij}^2)^{1/2} \qquad \ldots(3.16)$$

as a measure of the round-off noise arising from the i^{th} equation

in (3.3). Because equation (3.4) provides the constraint that

$$\alpha_i = (1 - \sum_{j=1}^{k} a_{ij}) \qquad \ldots(3.17)$$

then, equation (3.16) becomes

$$N_i = ((1 - \sum_{j=1}^{k} a_{ij})^2 + \sum_{j=1}^{k} a_{ij}^2)^{1/2} \qquad \ldots(3.18)$$

We seek therefore to minimize N_i with respect to its arguments a_{ij}, $j=1,2,\ldots k$. Hence we require that

$$\frac{\partial N_i}{\partial a_{ij}} = 0 \quad \text{for } j=1,2,\ldots k \qquad \ldots(3.19)$$

which provides k additional equations. Repeating this process for each i in the range 1,2,...k yields k^2 equations. In this way, the $2k(k+1)$ equations required to uniquely specify the free parameters in (3.3) are obtained. In particular equation (3.19) provides the system of linear equations:

$$\begin{bmatrix} 2 & 1 & 1 & \ldots & 1 \\ 1 & 2 & 1 & \ldots & 1 \\ \vdots & & & & \\ 1 & 1 & 1 & \ldots & 2 \end{bmatrix} \begin{bmatrix} a_{i1} \\ a_{i2} \\ \vdots \\ a_{ik} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

The subtraction of successive pairs of equations yields $a_{i1} = a_{i2}$

$= \ldots = a_{ik}$ from which it follows that $a_{ij} = \frac{1}{k+1}$.

In other words, the entries of the matrix A are all equal to $1/(k+1)$. We refer to the resulting procedure as the equi-weight predictor (EWP) form of the BIPC approach. The predictor formulas for the EWP form for the cases k=2, 4, 6, and 8 are explicitly shown in Appendix B.

3.2.3.2 NWP Form

The impact of round-off errors becomes less significant as the internal precision of the number representation increases. Since modern microprocessors typically have coprocessors for double precision arithmetic (with only minor performance penalty with respect to single precision), it would seem more beneficial to use the additional k^2 free parameters for some purpose other than minimizing round-off errors. A meaningful alternate strategy for choosing the entries of the matrix A can therefore be based on improving the stability property of the formulas. Roughly speaking, the stability of a numerical method refers to the behaviour of the difference between the generated solution and the exact solution of a problem. It is required that this difference does not grow as more steps are taken. As is demonstrated in section 3.3, by setting the entries of the matrix A to zero, the resulting BIPC formulas have a stability characteristic that is superior to that based on the equi-weight predictor. We choose this as our second BIPC alternative and refer to it as the null-weight

predictor (NWP) form. The predictor formulas for the NWP form are explicitly given in Appendix B for the cases k=2,4,6, and 8. Note also that because the predicted values in the NWP form are obtained using only derivative values, its computational complexity is reduced with respect to the EWP form. This feature provides an additional important advantage of the NWP form.

3.3 STABILITY OF INTEGRATION METHODS

The stability property of a numerical method for solving ode's is concerned with the behavior of the global error $\varepsilon_j = y_j - y(t_j)$ where $y(t_j)$ is the true solution at time t_j. For obvious practical reasons, the desirable feature in this regard is that the global error remain bounded as $j \to \infty$. Unfortunately a study of the global error behaviour of a method cannot be decoupled from the specific problem to which the method is applied. Thus, in order to obtain a basis for comparing stability properties of different methods, it has become conventional to explore this important matter in the context of a specific test problem; namely,

$$\dot{y}(t) = c\, y(t), \quad c \text{ complex} \qquad \ldots(3.20)$$

Note that if $Re(c) < 0$ then any solution of (3.20) remains bounded.

In general terms, a numerical method is said to be absolutely

stable for a particular $\lambda = ch$ (h is the fixed stepsize used with the method) if it produces bounded approximations when applied to (3.20). The region of absolute stability is that region of the complex-λ plane which is defined by the λ values for which the method is absolutely stable (Baker & Phillips [1981]).

More specifically, the procedure for investigating stability begins by applying the formulas of the solution method under investigation to (3.20). This gives rise to a "stability polynomial", $\pi(\mu;\lambda)$ which is characteristic of the method. The behavior of the global error depends on the location of the roots $\mu_1, \mu_2, \ldots$ of this stability polynomial.

This, in particular, gives rises to the following definition [Hall & Watt (1976)]: A numerical method for ode's is absolutely stable for a given $\lambda = ch$ if, for that λ, all the roots of the stability polynomial lie within the unit circle. A region $\mathbb{R}$ of the complex-λ plane is a region of absolute stability of the method if the method is absolutely stable for all $\lambda \in \mathbb{R}$.

A convenient measure for comparing the stability of different methods is to compare the values of the intercepts of their respective stability regions with the negative real axis in the λ-plane. The specific relevance of this intercept, H^*, is that it provides the means for determining the maximum allowable stepsize for the method to generate a stable solution to (3.20) when c is real and negative. This value, H^*, is frequently referred to as

the absolute stability boundary (or interval) of the method and it is this value that is typically quoted in the documentation for a method (e.g. $H^*=-2.78$ for fourth order Runge-Kutta methods).

Our interest is to establish the absolute stability boundary of both the EWP and NWP forms of the BIPC approach for a range of values of k. This objective requires the formulation from (3.2), (3.3) and (3.20) of an iterative formula that expresses the set of solution values in each new k-block as a function of the set of values in the previous k-block.

From the substitution of (3.20) in (3.2) and (3.3) we can obtain, respectively,

$$X = (P+\lambda Q)\, X^0$$

and

$$X^p = (R+\lambda S)\, X_0$$

where:

$$X = (y_0, y_1, y_2 \cdots y_k)^T$$

$$X_0 = (y_0, y_{-1}, y_{-2} \cdots y_{-k})^T$$

$$X^p = (y_0, y_1^p, y_2^p \cdots y_k^p)^T$$

$$P = \left[\begin{array}{c|c} 1 & 0 \\ \hline \bar{\alpha} & 0_{k\times k} \end{array}\right] \qquad Q = \left[\begin{array}{c|c} 0 & 0 \\ \hline \beta & D \end{array}\right]$$

$$R = \left[\begin{array}{c|c} 1 & 0 \\ \hline \alpha & \Lambda \end{array}\right] \qquad S = \left[\begin{array}{c|c} 0 & 0 \\ \hline \beta & B \end{array}\right]$$

and $\lambda = ch$. It then follows that:

$$X = (P+\lambda Q)\ (R+\lambda S)\ X_0 \qquad \ldots(3.21)$$

The sequencing of the data values in X and X_0 is, however, not consistent because the values in X are ordered in terms of increasing time values, while those in X_0 are ordered in terms of decreasing time values. In order to deal with this situation and obtain the desired iterative formula, we introduce the $(k+1) \times (k+1)$ matrix

$$\psi = \begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & \cdots & 1 & 0 \\ \vdots & & & & \\ 1 & 0 & \cdots & 0 & 0 \end{bmatrix}$$

Because $\psi\psi = I$, equation (3.21) can be written as

$$\begin{aligned} X &= (P+\lambda Q)\ (R+\lambda S)\ \psi\psi\ X_0 \\ &= (P+\lambda Q)\ (R+\lambda S)\ \psi\ \bar{X} \\ &= T\ \bar{X} \end{aligned}$$

where $\bar{X} = \psi X_0 = (y_{-k}, y_{-k+1}, \cdots y_{-1}, y_0)^T$ and $T = (P+\lambda Q)\ (R+\lambda S)\ \psi$.

Furthermore, note that X and $\bar{X}$ hold consistently ordered data values. This result therefore represents the desired transformation from old to new data values and the relevant property is characterized by the eigenvalues of the $(k+1)\times(k+1)$ matrix, T. More specifically, the absolute stability boundary is given by the largest H such that if $\lambda \in (-H,0)$ then all the eigenvalues of T have modulus less than one.

The eigenvalues, μ, of the T matrix are calculated from

$$\pi(\mu,\lambda) = \det(T-\mu I) = \sum_{i=0}^{k+1} e_i(\lambda)\,\mu^i = 0$$

One method used for establishing intervals of stability is the root locus method [Lambert 1973]. It consists of repeatedly solving this polynomial equation for a range of λ in the neighbourhood of the origin seeking the largest λ that ensures that all roots are less than one in magnitude. Any standard numerical method, such as Newton-Raphson iteration may be employed to find the zeros of the above polynomial; i.e. the eigen values of the matrix T. Our calculation, however, is based on using the IMSL library routine EIGRF to find the eigen values of the matrix T.

The procedure is as follows:

1. set $\lambda=0$ and $\Delta\lambda=-0.01$
2. call the EIGRF routine to get the eigenvalues μ_i $(i=1,2,\ldots,k+1)$ of the T matrix.
3. if $\max_{i=1,\ldots,k+1} \mu_i < 1$ then set $\lambda = \lambda+\Delta\lambda$ and go to 2 else the absolute stability boundary of the formula is found and it is equal to $\lambda-\Delta\lambda$.

Stability boundaries for both forms of the BIPC; i.e., the EWP and the NWP, have been calculated for the cases $k=2,3,\ldots,16$ and are given in Table (3.1). It should be noted the stability boundaries for the proposed NWP form are superior to those of the

Stability boundries

METHOD	EWP	NWP	NWP/EWP
k=2	0.439	0.576	1.31
k=3	0.311	0.326	1.05
k=4	0.179	0.222	1.24
k=5	0.149	0.168	1.13
k=6	0.088	0.135	1.53
k=7	0.070	0.144	1.63
k=8	0.042	0.098	2.33
k=9	0.029	0.087	3.00
k=10	0.018	0.078	4.33
k=11	0.0003	0.0007	2.33
k=12	0.015	0.067	4.46
k=13	0.020	0.061	3.05
k=14	0.007	0.055	7.85
k=15	0.0003	0.0006	2.00
k=16	0.008	0.011	1.38

Table 3.1 Stability Boundaries for the EWP and NWP Forms

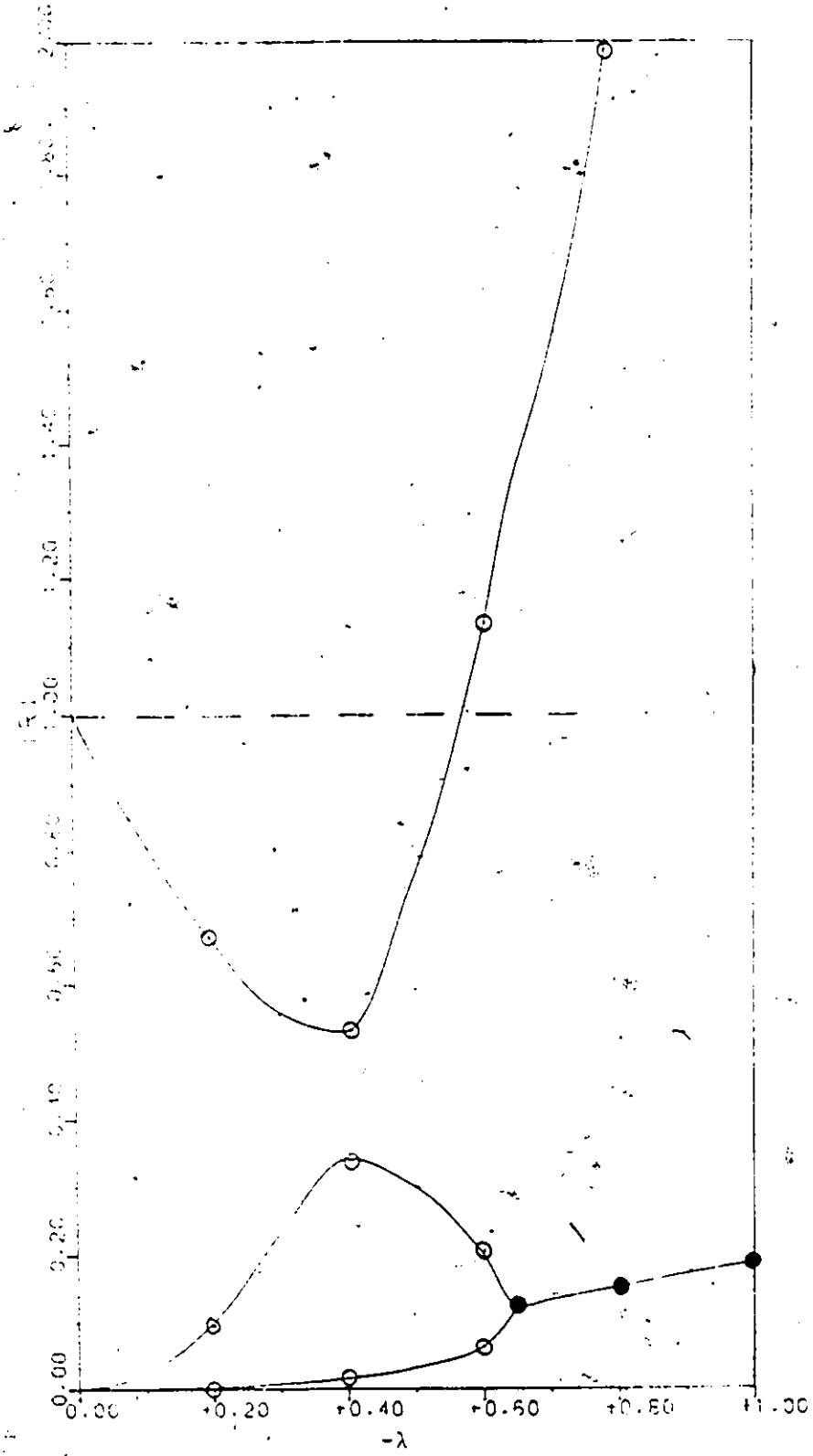

Figure 3.2: Roots of the NWP Form (k=2)

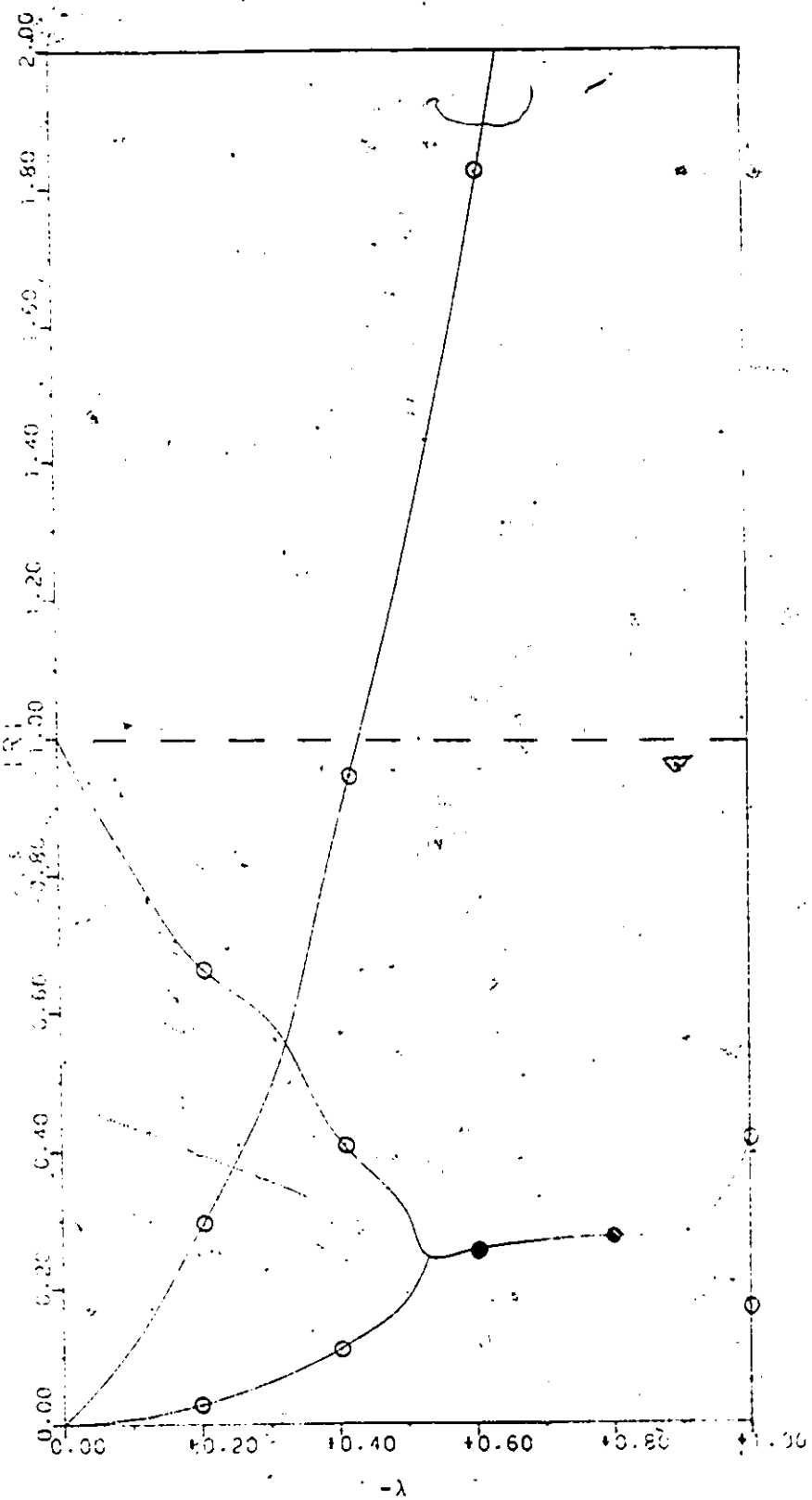

Figure 3.3: Roots of the EWP Form (k=2)

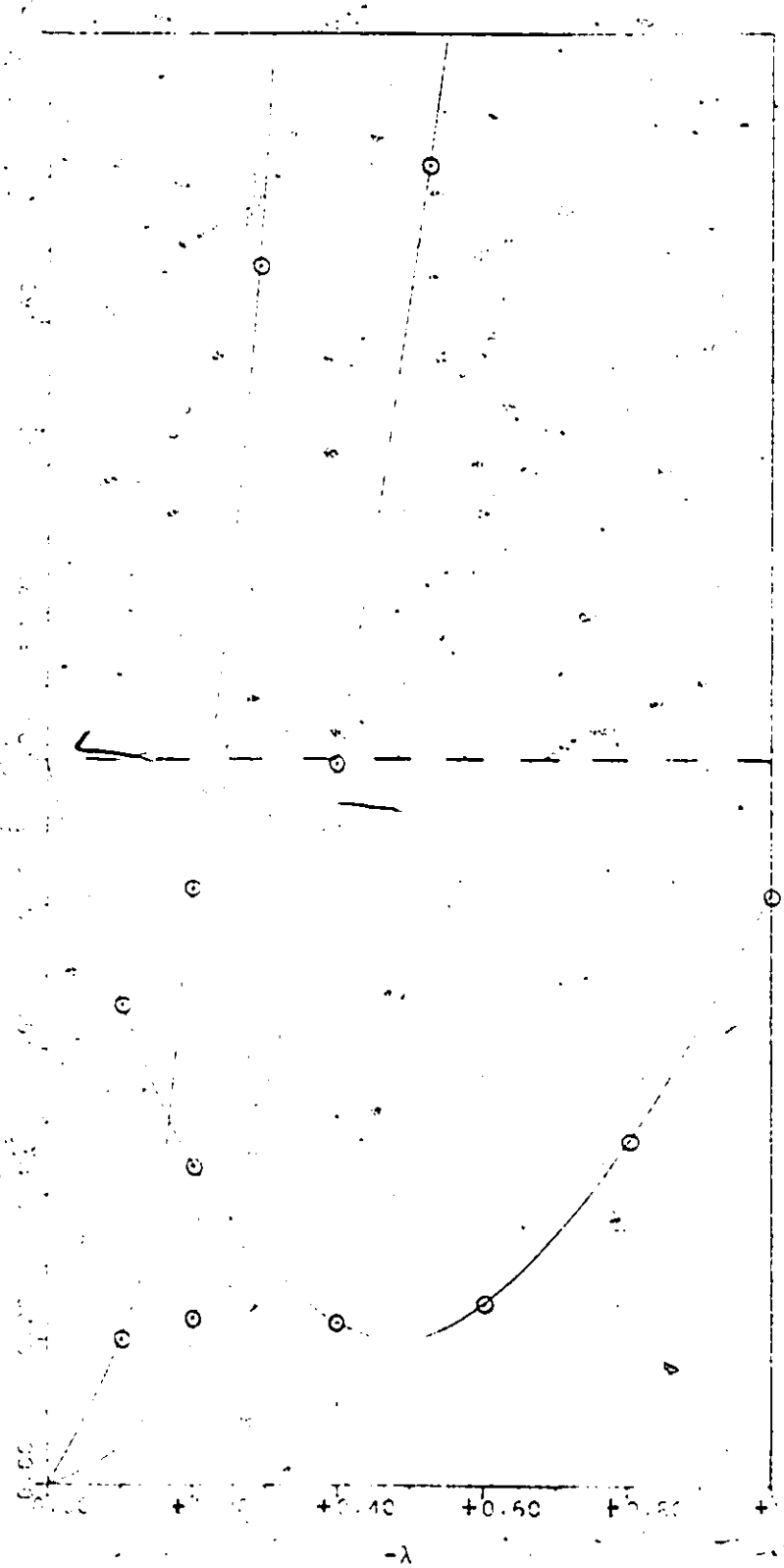

Figure 3.4: Roots of the NWP Form (k=4)

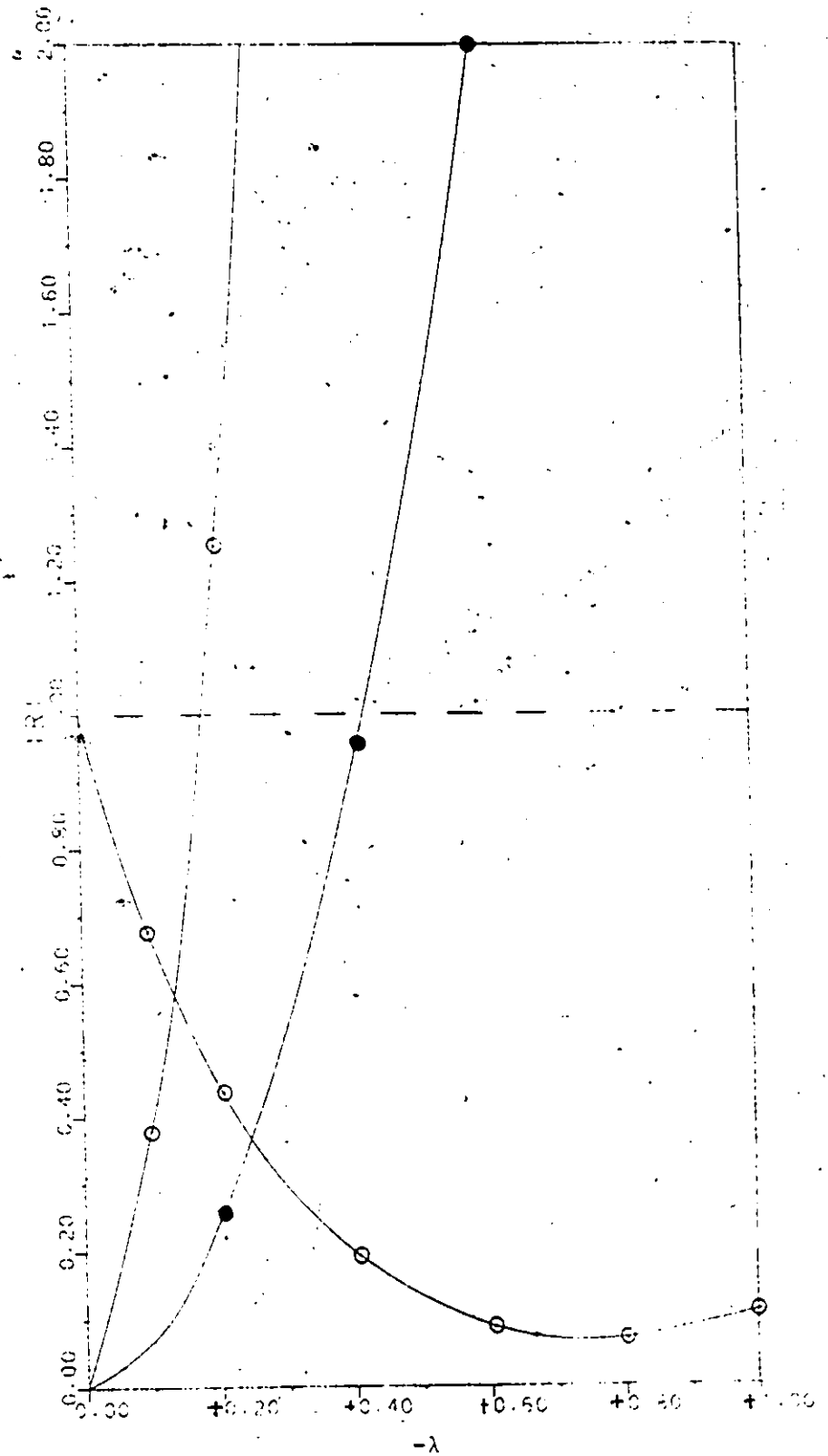

Figure 3.5: Roots of the EWP Form (k=4)

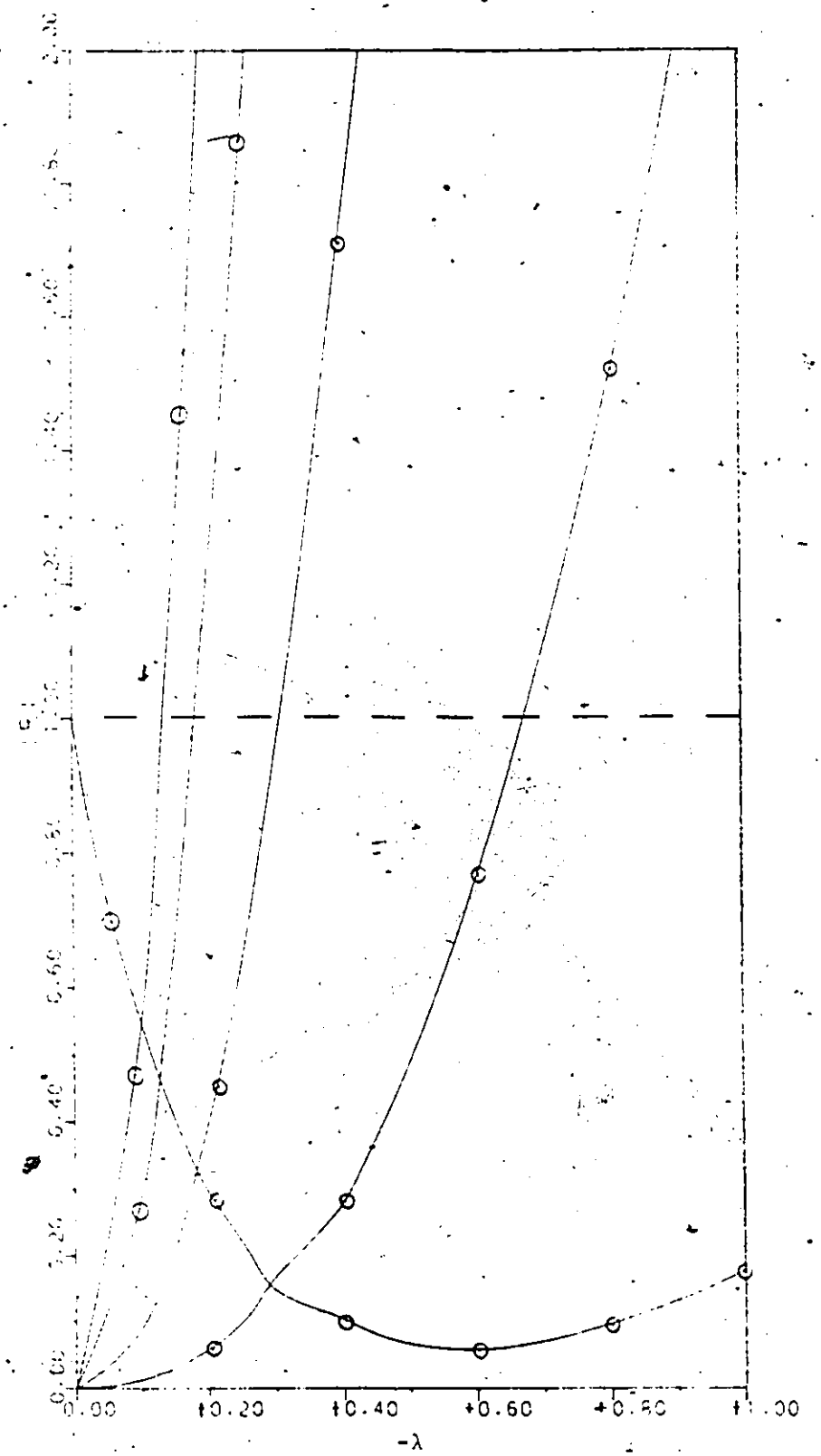

Figure 3.6: Roots of the NWP Form (k=6)

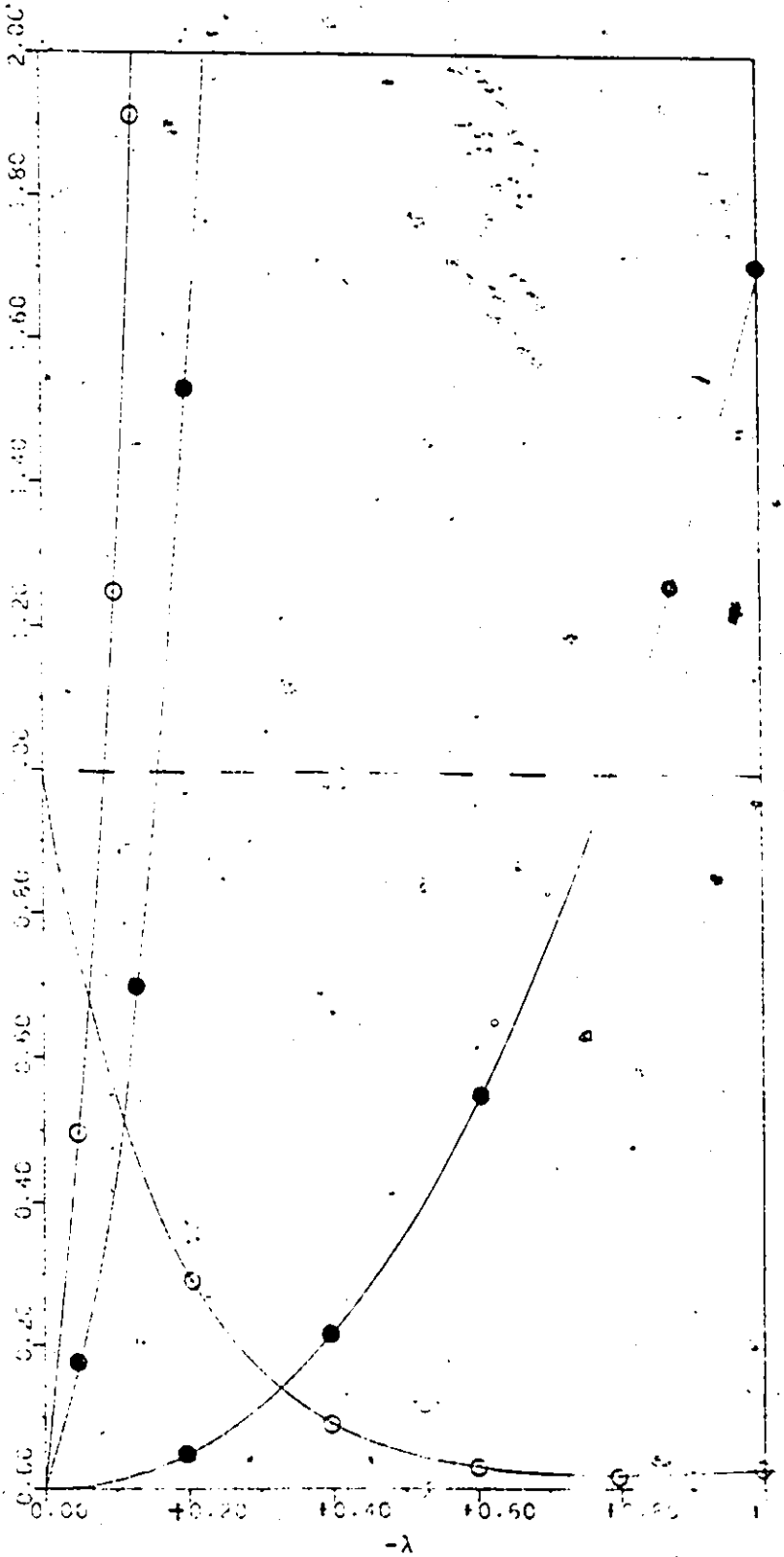

Figure 3.7: Roots of the EWP Form (k=6)

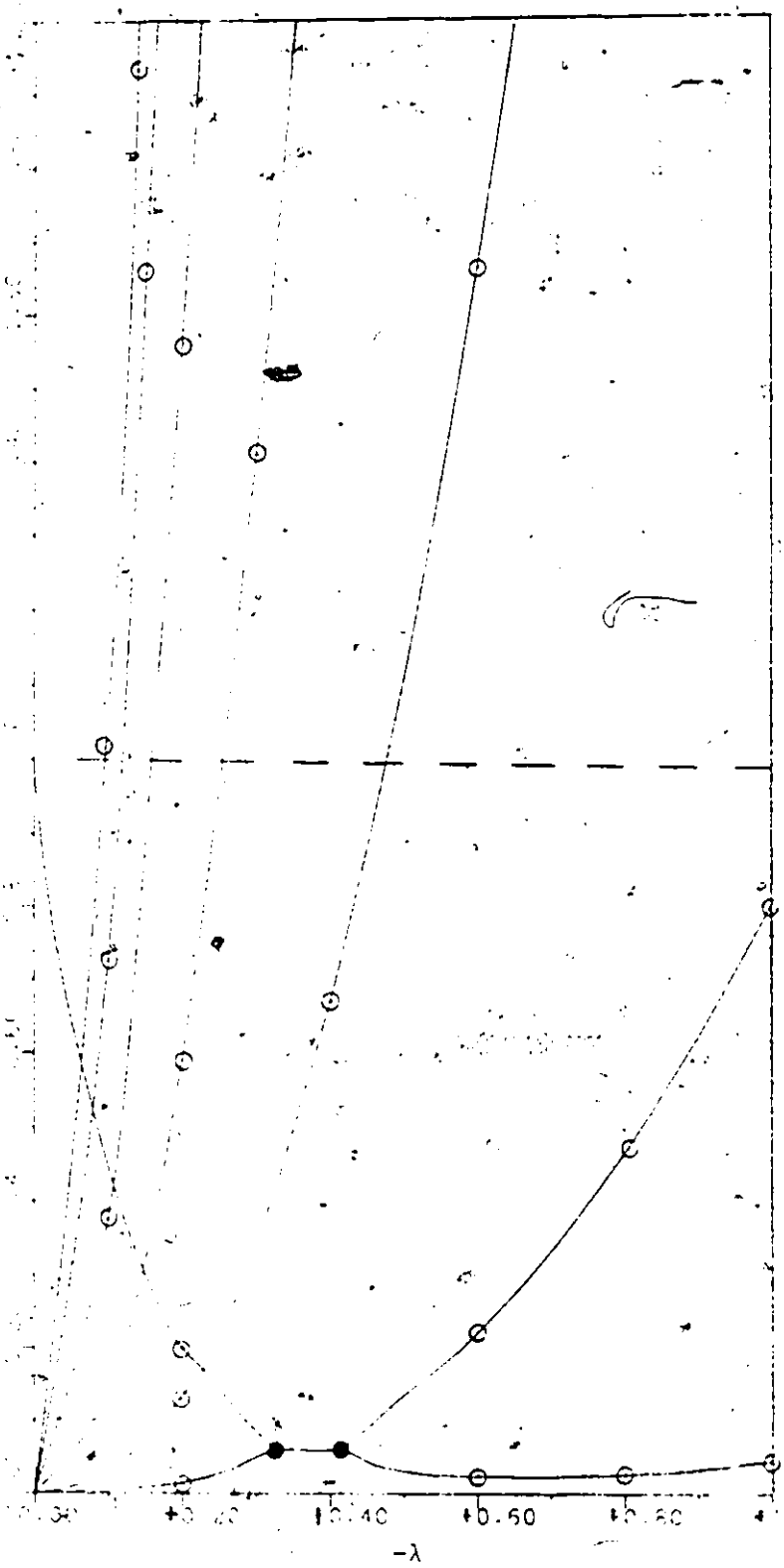

Figure 3. 8: Roots of the NWP Form (k=8)

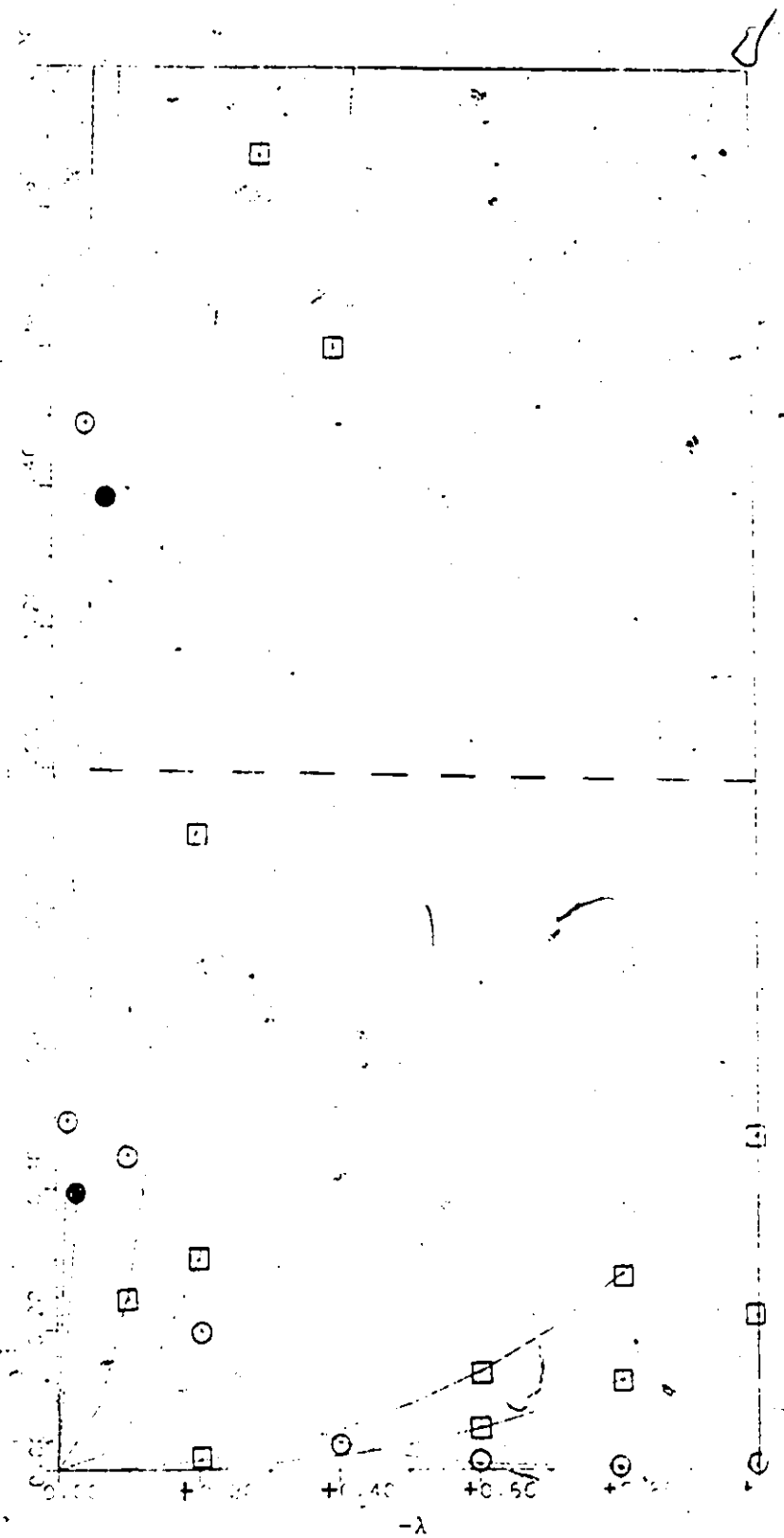

Figure 3.9 : Roots of the EWP Form (k=8)

EWP form by factors ranging from 1.05 for k=3 to 7.85 for k=14. The behaviour of the roots as a function of λ for both forms for the cases k=2,4,6, and 8 is shown in Figures (3.4-3.11). In these figures, positive roots are designated by a circle, negative roots by a square, and complex roots by a shaded circle.

3.4 STARTING THE BIPC COMPUTATION

One of the computational defects of predictor-corrector methods is that the methods are not self-starting. Predictor-corrector methods, in general, use the values of the dependent variable and its derivative at k different mesh points t_i, t_{i-1}, ...t_{i-k} to calculate the dependent variable at the next point(s). These methods are therefore commonly called multistep methods [Gear 1971]. Since we are given only y_I as an initial value, the other values required to start a predictor-corrector method must be computed in a preliminary step. To calculate the starting values, one-step methods are used; these require only the value at one mesh point to compute the value at the next. Probably the most widely used starting methods are those of the Runge-Kutta type. These methods are, however, serial in nature. Consequently, in a multiprocessor system, only one processor is active while the remainder are idle. In contrast, the starting method we propose is the block implicit one-step method [Shampine & Watts 1969; Worland 1976] which uses one mesh point to generate k points simultaneously in a parallel manner. Hence, the workload can easily be assigned to k processors. The block implicit one-step

method consists of Euler's formula (3.22) used as a predictor and the corrector equations (3.2) of the BIPC method outlined earlier. The formulas for the two-point block scheme are:

$$y_{r,0} = y_I + rhf_I \quad ; r=1, \ldots k \qquad \ldots(3.22)$$

$$y_{1,s+1} = y_I + \frac{h}{12}(5f_I + 8f_{1,s} - f_{2,s}) \qquad \ldots(3.23)$$

$$y_{2,s+1} = y_I + \frac{h}{3}(3f_I + 4f_{1,s} + f_{2,s})$$

where s=0,1...

The corrector equations (3.23) are iterated to convergence, i.e., until

$$(y_{i,s+1} - y_{i,s}) \leq \tau_r y_{i,s+1} + \tau_a \quad , \quad i=1,2$$

where τ_r and τ_a are the user specified relative and absolute tolerance errors. If the stepsize is small enough, convergence usually occurs after 1 or 2 iterations. The k starting values generated in this way then make it possible to initiate the BIPC method.

3.5 EXTENSION TO VARIABLE STEPSIZE

Once the method for starting the solution is used, BIPC methods can be evoked and the solution continued as far into the t domain as desired. However, in the second and major portion of the computation, a most important parameter is the stepsize to

use. In order to be reasonably effective and efficient, a numerical integration method should have the capability of adapting to variations in the local truncation error which occurs along the solution trajectory. This adaptation is typically achieved by a "scheme" for selecting the integration stepsize, h, in a way which ensures that this error becomes neither unacceptably large nor unduly small (thereby resulting in wasted computing effort), and a "technique" for handling these variable steps. In other words, an integration method is the combination of three parts: a particular formula, a scheme, and a technique.

3.5.1 Schemes for Stepsize Selection

An effective mean for establishing the stepsize to control the error in the integration process is essential if the procedure is to progress efficiently and reliably. Strategies for achieving variations in the integration stepsize are typically based on a solution quality criterion expressed in terms of the ratio $R_i = \left|\frac{\varepsilon_i}{\hat{\varepsilon}_i}\right|$ where ε_i is an estimate of the local truncation error associated with the solution value y_i, and $\hat{\varepsilon}_i$ is an admissible user specified local truncation error tolerance.

Inasmuch as BIPC formulas simultaneously generate a block of k new solution values on each step (i.e., y_i, i=1,2,...k), it follows that whenever $R = \max(R_i)$ is less than or equal to unity, then the step can be considered to be successful, while if R is greater than unity, the step must be rejected and repeated with a

reduced integration stepsize. From an analysis of local error behaviour [see, for example, Shampine & Watts 1976], it can be demonstrated that the "best" stepsize for the next iteration (independent of whether R is greater than or less than unity) to achieve the required tolerance is:

$$h_{new} = \sigma\, h_{old} \qquad \text{with } \sigma = \left|\frac{1}{R}\right|^{\rho+1} \qquad \ldots(3.24).$$

where ρ is the order of the formula. For the RIPC formulas, $\rho=k+1$ for the internal points of the block and $k+2$ for the last point.

Throughout our considerations, the admissible local truncation error tolerance $\hat{\varepsilon}_i$ at t_i is chosen to have the form

$$\hat{\varepsilon}_i = \tau_r \left|y_i\right| + \tau_a$$

where τ_r and τ_a are user specified relative and absolute error tolerances, respectively. This approach provides a combined "relative + absolute" flavour for the admissible error parameter.

Local truncation error estimates with predictor-corrector methods typically have the form:

$$\varepsilon_i = C_i^* (y_i - y_i^p) \qquad \ldots(3.25)$$

where C_i^* is a constant and ε_i is the error estimate at t_i. A

conventional choice for C_i^* is unity and we refer to this approach as the "simple difference" approach.

An alternate choice for C_i^* can be developed based on the fact that the predictor and corrector formulas within the BIRC context are both of the same order, namely (k+1) for the k-block case (except for the corrector at t_k which is of order (k+2)). The principal local truncation errors for the predictors and correctors are respectively,

$$T_i^p = C_i^p \, h^{k+2} \, y^{[k+2]} \, (\zeta_i^p), \quad \zeta_i^p \in [t_{-k}, t_i] \quad i=1,\ldots,k-1 \qquad \ldots(3.26)$$

$$T_i = C_i \, h^{k+2} \, y^{[k+2]} \, (\zeta_i), \quad \zeta_i \in [t_0, t_i] \qquad \ldots(3.27)$$

where the values of C_i and C_i^p are given by (3.12) and (3.13) respectively.

It follows that the exact value of y at t_i, namely $y(t_i)$, can then be written as (with higher order error terms neglected)

$$y(t_i) = y_i^p + C_i^p \, h^{k+2} \, y^{[k+2]} \, (\zeta_i^p) = y_i + C_i \, h^{k+2} \, y^{[k+2]} \, (\zeta_i)$$

Thus

$$y_i - y_i^p = C_i^p \, h^{k+2} \, y^{[k+2]} \, (\zeta_i^p) - C_i \, h^{k+2} \, y^{[k+2]} \, (\zeta_i) \qquad \ldots(3.28)$$

Assuming now that the $(k+2)^{th}$ derivative is reasonably constant at the value ψ over the interval $[t_{-k}, t_k]$, then

$$y^{[k+2]}(\zeta_i^p) = y^{[k+2]}(\zeta_i) \qquad \text{—}$$

and furthermore, from (3.28), it follows that

$$y_i - y_i^p = (C_i^p - C_i)\, h^{k+2}\, \psi \qquad \ldots(3.29)$$

The interesting feature here is that equations (3.27) and (3.29) together provide a measure of the local truncation error ε_i at t_i in terms of y_i^p and y_i; namely,

$$\varepsilon_i = \frac{C_i}{C_i^p - C_i}\,(y_i - y_i^p) = T_i \qquad i=1,..,k-1 \qquad \ldots(3.30)$$

i.e., C_i^* in equation (3.25) is $C_i/(C_i^p - C_i)$.

The above equation represents a key result for implementing (3.24); i.e. for the determination of the next value of the stepsize. This technique is frequently referred to as Milne's device; however, we shall refer to it as the "weighted difference" approach.

A drawback of this approach within the context of BIPC methods is that equation (3.30) does not hold at the end point of the block (i.e., at $t = t_k$) because the predictor and corrector formulas at this point have different orders. Thus, the use of

the above procedure implies that the local truncation errors at only the internal points of the block are available for the determination of the stepsize while that at the end point is not taken into consideration.

3.5.2 Techniques for Handling Variable Steps

The step-control scheme defined by (3.24) may call for a reduction or permit an increase in the stepsize within the new block with respect to the previous block. However, it should be recalled that in the development of the predictor formulas in Section 3.2, there is an implicit assumption that the separation between the generated values in the new block is the same as for those in the previous block (back values). Thus, if the spacing is different; i.e., if the back values are separated by h and the new values by σh, then the previous formulation of the predictor formulas is no longer valid.

One technique is to generalize the coefficients, within the predictor formulas, so that the formulas properly accomodate stepsize changes between adjacent blocks. We outline this procedure, in the context of the EWP form, by first re-writing the i^{th} equation in (3.3) as:

$$y_i^p = \frac{1}{k+1}\left(\sum_{j=0}^{k} y_{-j}\right) + h\left(\sum_{j=0}^{k} b_{ij}\, f_{-j}\right) \qquad i=1,\ldots,k \qquad \ldots(3.31)$$

where $b_{i0} = \beta_i$.

From the fundamental theorem of calculus we have that

$$y(t_i) = y(t_{-j}) + \int_{t_{-j}}^{t_i} f(t,y)\, dt$$

and since

$$y(t_i) = \frac{1}{k+1} \sum_{j=0}^{k} y(t_i)$$

then

$$y(t_i) = \frac{1}{k+1} \sum_{j=0}^{k} \left(y_{-j} + \int_{t_{-j}}^{t_i} f(t,y)\, dt\right)$$

$$= \frac{1}{k+1} \sum_{j=0}^{k} \left(y_{-j} + \int_{t_{-j}}^{t_i} P(t)\, dt\right) \qquad \ldots(3.32)$$

where $P(t)$ is the polynomial of degree $\leq k$ which interpolates $f(t,y)$ at the points $t_{-k}, \ldots, t_0$.

The right hand side of (3.32) represents an approximation to the solution at t_i. Since this approximation and the one given by (3.31) are both based on a polynomial interpolation of $f(t,y)$ at $t_{-k}, \ldots, t_0$, and since there is only one such polynomial of degree k passing through these points [Shampine & Gordon 1975], it follows that both these approximations to $y(t_i)$ are the same; hence, the right-hand side of (3.32) is equal to y_i^p.

A representation for $P(t)$ is given by the Lagrange form; i.e.,

$$P(t) = \sum_{r=0}^{k} f_{-r} * \prod_{\substack{n=0 \\ n\neq r}}^{k} \frac{(t - t_{-n})}{(t_{-r} - t_{-n})} \qquad \ldots(3.33)$$

With the incorporation of the change of variable $sh = t - t_0$ (where h is the stepsize of the current block extending from t_{-k} to t_0) we obtain:

$$P(sh + t_0) = \sum_{r=0}^{k} f_{-r} * v_r(s)$$

where

$$v_r(s) = \prod_{\substack{n=0 \\ n\neq r}}^{k} \frac{(s+n)}{(n-r)}$$

If we let σh be the stepsize on the new block extending from t_0 to t_k, then:

$t = t_i$ implies that $s = (t_i - t_0)/h = i\sigma h/h = i\sigma$

and $t = t_{-j}$ implies that $s = (t_{-j} - t_0)/h = -jh/h = -j$

Thus

$$\int_{t_{-j}}^{t_i} P(t)dt = h^* \int_{-j}^{i\sigma} \sum_{r=0}^{k} f_{-r} * v_r(s)\, ds$$

$$= h^* \sum_{r=0}^{k} f_{-r} * \int_{-j}^{i\sigma} v_r(s)\, ds$$

$$= h^* \sum_{r=0}^{k} f_{-r} * (V_r(i\sigma) - V_r(-j))$$

where $V_r(s) = \int v_r(s)\, ds$

Equation (3.32) then becomes

$$y_i^p = \frac{1}{k+1} \sum_{j=0}^{k} (y_{-j} + h * \sum_{r=0}^{k} f_{-r} (V_r(i\sigma) - V_r(-j)))$$

$$= \frac{1}{k+1} \sum_{j=0}^{k} y_{-j} + \frac{h}{k+1} \sum_{j=0}^{k} \sum_{r=0}^{k} f_{-r} \left(V_r(i\sigma) - V_r(-j)\right)$$

$$= \frac{1}{k+1} \sum_{j=0}^{k} y_{-j} + \frac{h}{k+1} \sum_{r=0}^{k} f_{-r} \sum_{j=0}^{k} \left(V_r(i\sigma) - V_r(-j)\right)$$

$$= \frac{1}{k+1} \sum_{j=0}^{k} y_{-j} + h^* \sum_{r=0}^{k} f_{-r} * \left(V_r(i\sigma) - \frac{1}{k+1} \sum_{j=0}^{k} V_r(-j)\right). \quad \ldots(3.34)$$

Thus, the b_{ij} coefficients of equation (3.31) are

$$b_{ij} = V_j(i\sigma) + \theta_j \qquad \begin{matrix} i=1,\ldots,k \\ j=0,1,\ldots,k \end{matrix} \quad \ldots(3.35)$$

where $\theta_j = \frac{-1}{k+1} \sum_{r=0}^{k} V_j(-r)$

With these σ dependent coefficients, the predictor equations of the EWP form are generalized to accomodate variations in the integration stepsize between adjacent blocks. The specific variable stepsize predictor formulas, based on the above analysis are given in Appendix C for the cases k=2,4,6 and 8. Included also are expressions for principal local truncation errors.

The development of the variable stepsize predictor formulas for the NWP form approach closely parallels the above. Recall that in this case

$$y_i^p = y_0 + h * \left(\sum_{j=0}^{k} b_{ij} f_{-j}\right) \qquad i=1,\ldots,k \quad \ldots(3.36)$$

As a result, the equivalent of equation (3.34) is

$$y_i^p = y_0 + h * \sum_{r=0}^{k} f_{-r} * (V_r(i\sigma) - V_r(0))$$

But $V_r(0) = 0$, and therefore

$$y_i^p = y_0 + h * \sum_{r=0}^{k} f_{-r} * (V_r(i\sigma) \quad i=1,\ldots,k \quad \ldots(3.37)$$

Thus, the b_{ij} coefficients of equation (3.36) are

$$b_{ij} = V_j(i\sigma) \qquad \begin{matrix} i=,1,\ldots,k \\ j=0,1,\ldots,k \end{matrix} \quad \ldots(3.38)$$

By comparing equations (3.35) and (3.38), we notice that the only difference in the b_{ij} coefficients within the formulas given by (3.31) and (3.36) is that those in (3.36) have $\theta_j = 0$. Consequently, the predictor formulas for the NWP form can be trivially obtained from those given in Appendix C by setting the θ_j's to zero and replacing the first term of the formulas (i.e., $\frac{1}{k+1} \sum_{j=0}^{k} y_{-j}$) with y_0. Note also that the corresponding local truncation error constants, C_i^p, can be obtained from those shown, by setting the constant terms (i.e., the ones independent of σ) to zero.

It is noted that the predictor formulas given in Appendix B (for fixed stepsize) are special cases of those given in Appendix C, and can be obtained by simply setting $\sigma=1$.

Since the corrector equation (3.2) uses only information on

$[t_0, t_k]$, in which the spacing is $h_{new} = \sigma h_{old}$, we can write

$$Y = y_0 + h_{new} (\beta f_0 + DF^p) \qquad \ldots(3.39)$$

In summary therefore, the variable stepsize implementation of the BIPC process is based on:

(i) equation (3.25) (with C_i^* set either to 1, or to the value $C_i/(C_i^p - C_i)$) which provides a measure of the local truncation error at $t=t_i$,

(ii) equation (3.24) which provides a value for the parameter σ which relates the stepsize within adjacent blocks,

(iii) the use of the σ-dependent coefficients within the predictor formulas as given by either equation (3.35) (the EWP form) or by equation (3.38) (the NWP form),

(iv) the use of equation (3.39) to produce the solution values.

A second technique for handling stepsize changes between adjacent blocks consists of generating an alternate set of back values that are spaced by σh. These alternate values can then be used directly with the fixed-stepsize predictor formulas (shown in Appendix B) to establish the predicted values within the new block.

In this approach, an alternate set of both solution values and derivative values must be produced in the case of the EWP form. These can both be acquired from an appropriate interpolation formula using the available back values. (The derivative values could equally well be obtained by direct evaluation using the interpolated solution values).

If Lagrange interpolation (see equation (3.33)) is used for this purpose, then $(k+1)^2$ operations (addition and multiplications) are required to generate one solution value or one derivative value. However, k back points have to be generated, and in a multiprocessor environment, one processor would typically be allocated to generate one point. Thus, the workload of each processor would consist of $2(k+1)^2$ operations to generate the solution and the derivative at one point. However, in the case of the variable stepsize predictor formulas given in Appendix C, the computation of the V_i coefficients at one point requires roughly $(k+1)^2$; operations i.e., half the computation required by interpolation approach.

In the case of NWP form, an alternate set of only derivative values would need to be generated via interpolation. Hence, only $(k+1)^2$ operations are needed for the interpolation which is equal to the number of operations required to compute the V_i coefficients within the variable stepsize predictor formulas.

As a conclusion, the variable stepsize formulas have less workload than interpolation for EWP form, and equal workload for the NWP form. Hence, the interpolation alternative will not be considered further.

3.6 THE USE OF A MODIFIER

An interesting alternative for the error estimate of (3.30)

is to use it to improve the local accuracy of the corrected values (an idea originally suggested by Hamming [1959]). The approach here is simply to add T_i to y_i, i=1,..., k-1, thereby achieving a result of order (k+2) at all the points of the block. Furthermore, in addition to modifying the corrector values, the predictor values can be modified in a similar manner. An estimate for the principal local truncation error of the predictor formula t_i can be deduced from (3.26) and (3.29); namely,

$$T_i^p = \frac{C_i^p}{C_i^p - C_i} (y_i - y_i^p)$$

This estimate cannot be used to improve the predicted value since, at the prediction stage, the corrected value y_i is not yet known. However, if it is assumed that the difference between the corrected and predicted values does not change violently from block to block or that the local truncation error remains approximately constant over two adjacent blocks, then we may write

$$T_i^p = \frac{C_i^p}{C_i^p - C_i} (y_i - y_i^p) \approx \frac{C_i^p}{C_i^p - C_i} (y_{i-k} - y_{i-k}^p) \qquad \ldots(3.40)$$

The predicted value can then be improved or modified by adding the term T_i^p to y_i^p using only information calculated previously.

On the basis, we may now summarize the overall calculation in the following steps:

(i) generate the predictor values using equation (3.3),

(ii) modify the predictor values by adding T_i^p specified by (3.40),

(iii) evaluate the derivatives at the "modified" predictor values,

(iv) generate the corrector values using equations (3.2) and the derivative values from (iii),

(v) modify the corrector values by adding T_i specified by (3.30),

(vi) evaluate the derivative at the modified corrector values.

We shall refer to this technique as the use of a "modifier". It is important, however, to realize that the use of a modifier robs us of the possibility of also using (3.30) for step-control purposes. We are therefore confined in this case to the use of the simple difference approach for this purpose.

Chapter IV

EXPERIMENTAL EVALUATION OF SEVERAL RIPC METHODS

4.1 INTRODUCTION

While theoretical measures of error and stability are necessary for comparing integration methods, no such comparison is complete without an experimental evaluation of the methods across a variety of test problems.

In Chapter 3, the basis for several parallel, variable stepsize integration methods was developed. Methods of this type have not previously appeared in the literature. The methods fall into two categories that are distinguished by the underlying formulas upon which they are based; namely, the EWP-based methods and the NWP-based methods. The formulas for the first case have their origin in the narrow context of the fixed stepsize investigations of Shampine and Watts [1969] which were carried out for the special cases of k=2. The formulas for the NWP-form are newly proposed in this thesis.

Our general goal in this Chapter is to:

(a) evaluate the relative performance of a selected group of parallel variable stepsize integration methods which have been derived from the consideration in Chapter 3.

(b) examine the performance of these methods relative to a group of widely used serial methods.

These evaluations are based on solutions obtained from a group of test problems. In these evaluations, the principal comparison measure we rely on, is a count of the number of function evaluations; i.e., evaluations of the derivative function required to obtain a solution over a prescribed interval.

More specifically, our objectives are to explore two particular issues. The first of these is to gain insight into the performance of the selected parallel methods in terms of the parameter k on which the order of the method depends (i.e., order = k+1). In the most straightforward system implementation, k also corresponds to the number of processors that are needed.

Ideally as the number of processor is increased, we would hope to have a corresponding increase in the solution rate as measured in terms of a decrease in the number of right-hand side evaluations per processor. This aspect of performance is explored in some detail in this chapter by formulating a variety of meaningful ratios that reflect on this behavior.

The second important aspect of our assessment is concerned with the performance of the selected parallel methods as a function of desired solution accuracy. In this regard, a wide range of accuracy requirements were considered. The relationship

between the order of the methods and their accuracy behavior is also investigated.

Before presenting the results obtained, we give, in the next section, a brief description of the set of test problems used in the evaluation process. This is followed, in section 4.3, by a description of the various BIPC methods to be compared, as well as of four, well known, integration methods used as reference methods. In section 4.4, the criteria used for comparing the different methods, is presented. We outline, in section 4.5, the framework within which the experiments were carried out. Finally, in section 4.6, the results of the evaluation process are presented and conclusions are drawn.

4.2 TEST PROBLEMS

Each of the problems is characterized by a set of differential equations, an initial condition, an initial time t_I and a final time t_N which indicates that the integration is to proceed over the interval from t_I to t_N. In order to achieve meaningful conclusions, in this comparison process, it is necessary that a reasonably extensive and comprehensive collection of test problems be examined. Furthermore, in order to facilitate the determination of solution errors, it is desirable to use only problems that have an available explicit analytic solution.

Our experimental work is based on a set of fourteen test problems which are specified in Appendix D. Included in this collection are both linear and nonlinear problems (problems TP1,

TP3, TP8 and TP9 are linear, while problems TP2, TP4, TP5, TP6, TP7, TP10-TP14 are nonlinear). In all cases the initial time, t_I, has the value zero. Ten of these problems have been selected from the extensive work by Hull et al [1972] on comparing numerical solution methods for ode's. All problems used have an explicit analytic solution.

4.3 METHODS TO BE COMPARED

In general, a method for the solution of ode's is viewed as being composed of three distinct parts; namely, a formula that specifies how a new solution values is obtained in terms of one or more past values, a scheme to select the integration stepsize on each successive step, and a technique for accommodating changes in integration stepsize on successive steps (with the class of single step methods, e.g. Runge-Kutta methods, the last of these is not relevant).

4.3.1 BIPC Methods

As was indicated previously in Chapter 3, the three basic ingredients of a method are a formula which generates the new solution values, a scheme for error control and stepsize selection and a technique to incorporate changes in stepsize on two successive steps. In the previous chapter some alternatives for each of these have been formulated in the context of the BIPC approach. In this section we identify the particular options that have been selected for experimentation, together with some justification for these choices.

The two underlying BIPC formulas, we consider, are the EWP form and the NWP form specified by equations (3.31) and (3.36) respectively, and by equation (3.2). As noted previously, the basis for the first of these lies in the work of Shampine and Watts [1969], while the second is a new proposal which, as will be demonstrated, exibits superior performance. We consider the four cases of k=2,4,6 and 8 in the investigation.

Alternates to these two BIPC formulas can be obtained by incorporating the modifier concept derived in section 3.6. In particular, we add to the predictor and corrector values the weighted difference truncation error estimates (see equations (3.30) and (3.29)) and thereby create an alternate BIPC formula. We refer to these formulas as the modified-EWP form and the modified-NWP form.

The scheme for stepsize selection is based on equation (3.24) which, in turn, depends on the choices made for the local truncation error tolerance $\hat{\varepsilon}_i$ and the approximation used for local truncation error, ε_i. A common choice for $\hat{\varepsilon}_i$ has the form

$$\hat{\varepsilon}_i = \tau_r |y_i| + \tau_a$$

where τ_r and τ_a correspond to relative and absolute error tolerances. This particular form provides a combined "relative" plus "absolute" format for the local truncation error tolerance. Although τ_r and τ_a can be assigned independent values, we impose

the simplifying constraint that $\tau_r = \tau_a = \tau$ in which case

$$\hat{\varepsilon}_i = \tau \ (|y_i| + 1)$$

The parameter, τ, is referred to in the sequel as the error control parameter.

Two choices were discussed for ε_i in the previous chapter, namely; the simple difference approach and the weighted difference approach. Experiments carried out with these two approaches showed that the former consistently gave superior performance, and therefore, the results reported here are all based, exclusively, on the simple difference approach.

The technique we use for handling variable steps consists of using the general variable stepsize predictor formulas shown in Appendix C. An alternate idea of limiting stepsize changes over two successive blocks to halving and doubling was also evaluated. This approach has a lower overhead because the coefficients of the formulas are constants (i.e. $\sigma = 0.5$ or $\sigma = 2$) and hence do not require repeated re-evaluation. This is in contrast to those of the general variable stepsize formulas which have to be evaluated each time σ changes value. However, our experiments showed that the general variable stepsize approach greatly improved the performance of the methods relative to the halving/doubling approach and therefore only results based on the variable stepsize approach are given.

In summary then, results are presented for four variable stepsize BIPC methods each of which uses the simple difference approach for estimating the local truncation error. These are:

EWP/N: the EWP form (no modifier),

EWP/M: the modified - EWP form,

NWP/N: the NWP form (no modifier),

NWP/M: the modified - NWP form.

4.3.2 Reference Methods

a) Runge-Kutta (RK) Methods:

Numerical reference solutions, for comparison purposes, were obtained, for the test problems, using three different Runge-Kutta methods, each of different order. All three are of the dual order type. One of the methods is based on the Zonnveld [1964] 4/5 formula while the other two are based on the 6/7 and 8/9 formulas developed by Verner [1978]. In each of these cases, a variable stepsize implementation was achieved using the local truncation error estimates embedded in the formulas together with the stepsize control scheme specified by (3.24). Also, in each case, the solution estimate provided by the higher order result was adopted. The resulting methods are referred to in the sequel as RK5 (fifth order), RK7 (seventh order) and RK9 (ninth order) respectively.

b) Shampine-Gordon/Adam (SGA) code:

In addition to the three Runge-Kutta methods, a multistep reference method is also considered. It is based on the Adams predictor-corrector formulas together with extensions incorporated by Shampine and Gordon [1975] which yield a variable order - variable stepsize method. The code used in our experiments has been taken from this reference.

The code starts by formulas of order one and then increases the order as more data becomes available, if it appears desirable. Besides enabling the code to be self-starting, this permits the control of the error within the generated solution values by varying the order as well as the stepsize. This makes this method efficient at all tolerances.

The requisite error estimates come from comparing results at different orders; this requires no extra function evaluations. However, the tasks of selecting and changing the order and stepsize are relatively complex and add a high overhead to this code.

4.4 COMPARISON CRITERIA

The time taken for solving a differential equation may be divided into two parts; the time, T_d, taken to repeatedly evaluate the derivative function, f, and the overhead which is the time actually spent in solving the problem after excluding T_d.

Execution time measurements are complicated by the fact that mainframe computers are typically "time shared" so that the time spent by a program cannot be measured accurately. For problems with complex derivative functions, the time T_d dominates the overall solution time. Thus, a count of the number of evaluation of f is directly related to T_d and hence is a reasonable measure of the overall solution time. This approach also has the advantage of being independent of the type of computer used. For problems with simple derivative functions, the overhead time does make a significant relative contribution to the overall solution time. Nevertheless, because of the problem of accurate execution time measurement, we use the count of derivative function (right-hand-side) evaluations as our principal comparison criteria.

Another important issue is the error to be measured. Integration methods attempt to control the local truncation error. Hull et al [1972] define the local error by

$$\left| Y_J - Y(t_J; Y_{J-1}) \right|$$

where Y_J is the solution value generated by the method at t_J and $Y(t_J;Y_{J-1})$ is the true solution of the differential equation which passes through the last computed value. In their study, they measured how well some methods succeeded in controling this error by reporting the number of times the local error, produced by a method, exceeded a user specified tolerance τ.

However, users of methods for solving ode's, generally expect that when given a tolerance, τ, and a problem, the method will yield global, or true, errors of about τ. This error is defined by

$$\left| Y_J - Y(t_J;\ Y_I) \right|$$

where $Y(t_J;\ Y_I)$ is the true solution which passes through the given initial point. Shampine et al [1976] and Krogh [1973] have expressed the view that because of the user's concern with global error, this quantity should be emphasized. We share this view and use global error consideration throughout our evaluation procedure.

We impose a global error tolerance requirement, G_T, and for each experiment, the error tolerance parameter, τ, is suitably adjusted in a preliminary iterative phase in order to achieve solutions consistent with G_T.

4.5 EXPERIMENTAL FRAMEWORK

Our interest is to conduct experiments which establish how well various BIPC methods behave when providing a solution within a predefined global error requirement. An experiment consists of solving a test problem with a particular method at a specific error tolerance requirement, G_T. We define the global error which occurs along the solution trajectory as

$$G = \max_i G_i$$

where G_i is the global error in the i^{th} block. We define G_i as

$$G_i = \max_{1<j<k} \{q_j\}$$

where

$$q_j = \frac{y_j - y(t_j)}{n_j}$$

$$n_j = \begin{cases} |y_j| & \text{for } |y_j| > 1 \\ 1 & \text{Otherwise} \end{cases}$$

In the case of test problems with more than one differential equation, q_j is appropriately extended over all state variables.

If G is within a factor of 2 of the global error tolerance requirement, G_T, the experiment is considered successful and the number of right-hand-side evaluations (RHS) is recorded. Otherwise, the error tolerance parameter, τ, is adjusted, and the experiment is repeated. This process continues until a successful result is obtained. The experiments were carried out for three separate values of G_T; namely, 10^{-3}, 10^{-6} and 10^{-9}. This wide range was selected in order to reasonably evaluate a method's performance over different accuracy requirements.

A point to be noted here is that in the experiments no

restrictions have been placed on the stepsize. We also attempt to supress as much as possible the effect of a poor choice of a starting stepsize. We provide the methods with an arbitrary starting value ($h_{start} = (t_N - t_1)/200$) and use the starting procedure (of section 3.4) together with a stepsize adjustment procedure based on equation (3.24) to adjust this value until one is found for which $0.5\ G_T < G_1 < 2\ G_T$; i.e. the global error in the first block is within the prescribed bounds. The number of RHS evaluations during this process is not taken into account. The solution procedure then continues with the selected BIPC method which starts with this particular stepsize.

4.6 NUMERICAL RESULTS AND CONCLUSIONS

In this section, the results of applying the four BIPC methods, and the four reference methods, over the fourteen test problems, are presented. All computations were carried out on an Amdahl 470/V7 using double precision arithmetic.

For each method and each test problem, results for three global error tolerance specifications; namely, 10^{-3}, 10^{-6} and 10^{-9}, were obtained. For each of these, we report the number of RHS evaluations per processor. This is defined to be equal to:

$$\frac{\text{(total number of RHS evaluations in the single processor mode)}}{N_P}$$

where $N_p = 1$ for the reference methods and $N_p = k$ for the BIPC methods. This convention reflects the fact that the BIPC methods are intended to be used in a k processor environment where the total workload would be equally shared. The maximum global error that occurs along the solution trajectory is not reported because this error is always within a factor of 2 of the global error tolerance specification (due to the initial tuning step referred to in section 4.5) and we regard variations of this amount to be inconsequential. The results for the four BIPC methods, for each of the test problems, are shown in Tables (4.1 - 4.4). The results for the Runge-Kutta methods are given by Table (4.5) while those for the SGA code are shown in Table (4.6).

We first examine the data in these tables from the point of view of assessing the relative performance among the four BIPC methods under consideration. In particular, from an examination of the total number of RHS evaluations per processor, given in the last row of Tables (4.1 - 4.4), we note that the use of the modifier improves performance (i.e. provides a lower total number of RHS evaluations per processor) in the two lower order cases (k=2 and k=4). In other words, for these two cases, methods NWP/M and EWP/M give better performance than methods NWP/N and EWP/N respectively. This is true over all accuracies; i.e., all three values of the global error tolerance, G_T, and furthermore, the higher the accuracy (i.e., the smaller the G_T value) the greater is the improvement. We believe that the reason for this greater

TP \ GT	k=2			k=4			k=6			k=8		
	10^{-3}	10^{-6}	10^{-9}	10^{-3}	10^{-6}	10^{-9}	10^{-3}	10^{-6}	10^{-9}	10^{-3}	10^{-6}	10^{-9}
1	134	252	842	150	168	250	198	204	230	294	306	314
2	38	158	746	36	90	182	42	68	130	60	70	104
3	146	786	4216	116	264	688	84	180	348	100	136	216
4	26	92	444	22	46	110	24	40	66	26	32	48
5	470	2636	14772	154	482	1692	92	212	580	126	140	248
6	512	1684	13328	130	440	1360	136	176	416	206	216	318
7	740	4000	11850	216	540	1214	126	200	372	122	184	230
8	518	2826	15972	126	524	1642	136	214	582	210	214	290
9	128	766	4210	130	226	794	182	188	290	278	290	298
10	332	1300	6130	142	426	1330	128	234	450	178	190	484
11	354	2348	13178	238	588	1640	146	324	654	182	250	464
12	498	3560	19782	292	674	1954	226	424	872	222	350	684
13	702	4866	27334	386	888	2582	266	488	1166	254	440	808
14	1326	8348	49570	564	1466	4794	576	1166	2190	382	850	2820
TOTAL =	5924	33622	182374	2702	6822	20232	2362	4118	8346	2640	3668	7326

TABLE 4.1 Number of RHS Evaluations Per Processor for Method EWP/N

	k=2			k=4			k=6			k=8		
TP \ GT	10^{-3}	10^{-6}	10^{-9}	10^{-3}	10^{-6}	10^{-9}	10^{-3}	10^{-6}	10^{-9}	10^{-3}	10^{-6}	10^{-9}
1	138	308	1310	188	200	372	402	450	462	488	596	618
2	32	154	1042	34	90	220	52	72	130	128	176	194
3	174	780	5242	128	238	714	108	172	344	168	244	330
4	26	108	490	22	42	102	26	40	60	26	38	54
5	282	786	4388	158	440	892	150	204	384	168	206	330
6	428	1678	6656	166	436	1078	260	296	410	376	404	494
7	504	1420	4460	214	308	1214	144	180	358	198	240	358
8	366	1424	9486	158	420	1166	250	274	494	328	358	394
9	158	578	2460	200	244	662	332	342	376	422	468	482
10	330	1298	5150	124	340	942	232	256	430	286	308	332
11	298	1396	4656	188	492	1036	246	320	652	350	352	542
12	318	1762	6990	292	646	1384	254	424	860	384	460	734
13	648	2734	10858	286	680	2582	290	484	1070	445	660	834
14	1324	4960	41630	470	1362	3174	576	1070	2152	524	1202	1488
TOTAL =	5026	19386	104818	2628	5938	15538	3322	4584	8182	4291	5712	7184

Table 4.2 Number of RHS Evaluations Per Processor for Method EWP/M

	k=2			k=4			k=6			k=8		
TP \ GT	10^{-3}	10^{-6}	10^{-9}	10^{-3}	10^{-6}	10^{-9}	10^{-3}	10^{-6}	10^{-9}	10^{-3}	10^{-6}	10^{-9}
1	104	186	766	126	146	230	136	146	180	144	148	158
2	36	158	746	36	90	180	38	66	130	46	64	104
3	146	786	4214	116	268	702	78	168	348	80	132	216
4	26	92	444	22	46	110	24	40	66	26	32	48
5	470	2636	14772	162	486	1692	94	230	580	78	128	248
6	510	1684	13326	140	440	1360	118	206	416	110	140	326
7	740	3998	11850	216	540	1216	124	200	372	112	182	238
8	516	2824	15972	142	524	1642	110	222	582	112	152	290
9	132	770	4210	118	226	794	134	138	290	138	148	176
10	332	1300	6130	142	426	1330	128	234	450	112	184	228
11	354	2348	13178	238	588	1640	142	324	654	150	250	464
12	498	3560	19782	292	674	1954	226	424	872	206	350	684
13	702	4866	27334	386	888	2582	266	428	1166	254	440	808
14	1326	8348	49570	564	1466	4794	576	1166	2190	378	834	2728
TOTAL =	5892	33556	182294	2700	6806	20226	2194	3992	8296	1946	3184	6716

Table 4.3 Number of RHS Evaluations-Per Processor for Method NWP/N

TP \ GT	k=2			k=4			k=6			k=8		
	10^{-3}	10^{-6}	10^{-9}	10^{-3}	10^{-6}	10^{-9}	10^{-3}	10^{-6}	10^{-9}	10^{-3}	10^{-6}	10^{-9}
1	114	284	1296	140	156	284	152	162	188	164	170	180
2	32	154	706	32	90	178	38	66	128	50	54	104
3	174	786	3124	112	238	660	84	164	346	94	152	214
4	26	82	464	22	42	102	26	40	60	26	36	50
5	282	786	4394	158	440	892	88	198	384	98	132	236
6	428	1680	6666	166	436	1078	104	184	410	114	122	178
7	512	1390	4466	214	308	1214	118	176	358	112	186	250
8	366	1424	7988	150	420	1166	144	224	492	152	174	244
9	146	578	2292	162	224	662	184	190	290	198	202	212
10	330	1300	5156	126	340	942	128	224	426	118	186	218
11	298	1398	4664	188	492	1036	158	314	652	158	248	496
12	318	1764	6998	292	646	1334	196	410	856	194	334	608
13	614	2738	10872	270	680	2582	276	484	908	254	440	816
14	1326	4968	41684	504	1370	3174	572	1066	2148	350	724	1448
TOTAL =	4966	19332	100770	2536	5882	15304	2268	3902	7646	2082	3170	5254

Table 4.4 Number of RHS Evaluations Per Processor for Method NWP/M

TP \ GT	RK5			RK7			RK9		
	10^{-3}	10^{-6}	10^{-9}	10^{-3}	10^{-6}	10^{-9}	10^{-3}	10^{-6}	10^{-9}
1	117	187	530	167	218	385	330	379	523
2	56	105	273	70	130	250	96	160	224
3	278	1232	5629	216	676	1330	360	570	1153
4	54	137	428	59	117	207	79	127	206
5	440	1405	3897	391	700	1510	383	656	1280
6	488	1372	3897	550	1260	3339	640	1376	2960
7	393	1522	8212	300	859	2159	431	830	1679
8	661	2096	8342	346	928	1940	570	1143	2174
9	243	911	3107	199	529	1394	381	730	1509
10	399	1207	4688	541	1020	2459	586	1040	2224
11	597	1853	7332	576	1225	2859	662	1249	2271
12	792	2782	9187	672	1556	3189	849	1449	2612
13	1162	3522	13931	933	2085	4445	1177	1795	3835
14	1977	5926	23750	1486	3953	7637	1897	3820	6959
TOTAL =	7657	24257	93203	6506	15256	33103	8441	15324	29609

Table 4.5 Number of RHS Evaluations for Runge-Kutta Methods

TP \ GT	10^{-3}	10^{-6}	10^{-9}
1	95	152	238
2	39	127	245
3	169	298	520
4	31	71	142
5	178	408	711
6	261	410	714
7	261	457	661
8	251	424	772
9	179	383	692
10	209	385	764
11	311	657	964
12	416	864	1690
13	649	1158	2114
14	944	2422	4048
TOTAL =	3993	8216	14275

Table 4.6 Number of RHS Evaluations for the SGA Code

improvement, at high accuracy, is due to the fact that the use of the modifier increases the order of the methods and it is generally true that high order methods are more efficient at high accuracies than lower order methods. The fact that the use of the modifier also improved performance at low accuracy is somewhat counter intuitive to the generally accepted view that at low accuracies, low order methods are preferable.

For the two higher order cases (k=6 and k=8), the modifier improves performance at high accuracy ($G_T=10^{-9}$) while for low accuracy ($G_T=10^{-3}$), its use deteriorates performance. This is due to the same reasons mentioned above. For the EWP form with moderate accuracy solutions $G_T=10^{-6}$), the use of the modifier also deteriorates performance. For the NWP form with $G_T=10^{-6}$, the use of the modifier gives a marginal improvement. But because of the overhead associated with the use of the modifier (evaluation of the T_L and T_i^p as described in section 3.6), its use could not be recommended for k=6 and 8, when solutions of moderate accuracy are desired and it is therefore not the selected alternative in the discussion that follows.

By way of comparing the performance of the two methods based on the EWP form (i.e. EWP/N and EWP/M), we show in Table (4.7) the alternative which yields superior performance for each value of k and each of the three values of G_T (the total number of RHS evaluations per processor is also included for each case). Table (4.8) provides an analogous summary for the case of the two methods that are based on the NWP form.

k \ GT	10^{-3}	10^{-6}	10^{-9}
k=2	EWP/M 5026	EWP/M 19386	EWP/M 104818
k=4	EWP/M 2628	EWP/M 5938	EWP/M 15538
k=6	EWP/N 2362	EWP/N 4118	EWP/M 8182
k=8	EWP/N 2640	EWP/N 3668	EWP/M 7184

Table 4.7 Best EWP Performance

k \ GT	10^{-3}	10^{-6}	10^{-9}
k=2	NWP/M 4966	NWP/M 19332	NWP/M 100770
k=4	NWP/M 2536	NWP/M 5882	NWP/M 15304
k=6	NWP/N 2194	NWP/N 3992	NWP/M 7646
k=8	NWP/N 1946	NWP/N 3184	NWP/M 5254

Table 4.8 Best NWP Performance

It should be stressed here that the option of including the modifier or not has no impact on the multiprocessor realization (i.e., the number of processors). It can be achieved by a simple software switch.

We first note that in Tables (4.7) and (4.8) a desired feature is a monotonic decrease in the number of RHS evaluations per processor as k increases (for each value of G_T). With only one exception (EWP with $G_T = 10^{-3}$) the data does have the property. The implication in this exceptional case is that solution time, as measured by RHS evaluations per processor, increases when the number of processors is increased from 6 to 8, a situation which is clearly counterproductive.

Another interpretation of these results is that an eight processor (N_p=k=8) implementation of the NWP form of the RIPC process yields a system that is "globally superior"; i.e., unlike typical fixed-order methods, it provides superior performance over a wide range of accuracy requirements. This is not the case for the EWP form.

In order to explore this behaviour in more detail it is useful to examine the incremental advantage that results from an increase in the number of processors. Suppose, in particular, that k_a and k_b are two distinct values of k with $k_a = ck_b$ and $c < 1$ and that $(RHS)_a = c' (RHS)_b$. As noted above, a minimal requirement is that $c' > 1$. However a far more desirable condition is that $cc' > 1$ since this ensures not only an absolute,

but also a relative advantage of the additional processors. In Tables (4.9) and (4.10), values of c' and cc' are shown for adjacent values of k_a and k_b using data from Tables (4.7) and (4.8) respectively. As expected, the values of c' (first entry in each tuple) are consistently greater than one, except for the one case identified above. However the values of the product cc' (second entry in each tuple) are greater than one in fewer than half the cases and almost all of these occur in the high accuracy tests ($G_T = 10^{-9}$). For moderate accuracy solutions, some of the values are greater than one and the others are only slightly less than one and are still considered to be acceptable. The results for low accuracy solutions suggest the existence of a "diminishing returns" phenomenon.

An interesting alternate presentation of the results in Tables (4.1-4.4) is given in Figure (4.1). Here the BIPC method which gives the lowest total number of RHS evaluations per processor, at each k and G_T, is given together with the percentage performance deterioration of the other BIPC methods relative to the best method.

From this Figure we note the general superiority of the NWP form relative to the EWP form since an NWP entry appears at the head of each column. The improvement in performance becomes more significant as k increases. This could possibly be because the ratio of the stability boundries for the NWP formulas relative to the EWP formulas generally increases as k increases (see Table 3.1)). We also observe from the Figure that for k=2 and 4, the

$k_a + k_b$ \ G_T	10^{-3}	10^{-6}	10^{-9}
2 + 4	(1.91, 0.96)	(3.26, 1.63)	(6.75, 3.38)
4 + 6	(1.11, 0.74)	(1.44, 0.96)	(1.90, 1.27)
6 + 8	(0.89, 0.67)	(1.12, 0.84)	(1.14, 0.86)

Table 4.9 Incremental Advantage for the EWP Form

$k_a + k_b$ \ G_T	10^{-3}	10^{-6}	10^{-9}
2 + 4	(1.96, 0.98)	(3.29, 1.65)	(6.58,3.29)
4 + 6	(1.16, 0.77)	(1.47, 0.98)	(2.00, 1.33)
6 + 8	(1.13, 0.85)	(1.25, 0.94)	(1.46, 1.10)

Table 4.10 Incremental Advantage for the NWP Form

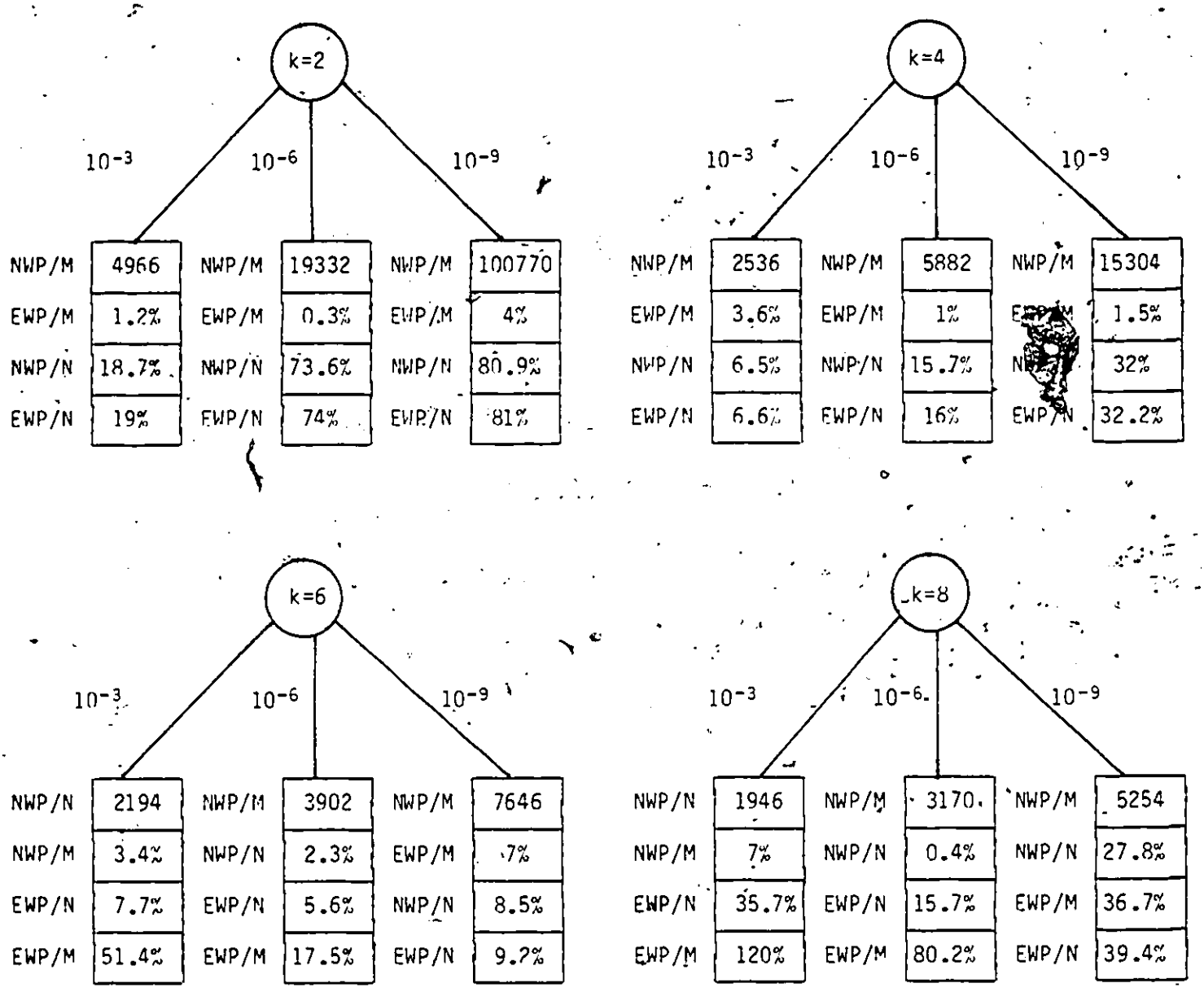

Figure 4.1 Relative Performrance Among the BIPC Methods

second entry is an FWP form while for k=6 and 8, it is an NWP form (except for one case; namely, k=6, $G_T=10^{-9}$). The percentage deterioration of the second entry is relatively minor, (except for the case k=8, $G_T=10^{-9}$). The substantial benefit of using the modifier at k=2 and 4 is also apparent from the Figure.

Our comparison of the performance of the BIPC concept with the results obtained from the reference methods is confined to the superior case of the NWP form (i.e.; the results shown in Table (4.8)). The corresponding RHS evaluations per processor over the test problems are shown in Table (4.11). The comparison is presented in terms of two separate "speed-up ratios". Both of these reflects average relative performance over all test problems, but they differ in the way in which this average behaviour is formulated.

In the first case, which we denote by S_1^{RK} or S_1^{SGA} (depending on whether the comparison is with respect to Runge-Kutta methods or the SGA code) we compare total RHS evaluation over all test problems by forming the ratios

$$S_1^{RK}(k,G_T) = \frac{RK(G_T)}{NWP(k,G_T)}$$

$$S_1^{SGA}(k,G_T) = \frac{SGA(G_T)}{NWP(k,G_T)}$$

for each value of k and G_T. The values of $NWP(k,G_T)$ are taken

TP \ GT	k=2			k=4			k=6			k=8		
	10^{-3}	10^{-6}	10^{-9}	10^{-3}	10^{-6}	10^{-9}	10^{-3}	10^{-6}	10^{-9}	10^{-3}	10^{-6}	10^{-9}
1	114	284	1296	140	156	284	136	146	188	144	148	180
2	32	154	706	32	90	178	38	66	128	46	64	104
3	174	786	3124	112	238	660	78	168	346	80	132	214
4	26	82	464	22	42	102	24	40	60	26	32	50
5	282	786	4394	158	440	892	94	230	384	78	128	236
6	428	1680	6666	166	436	1078	118	206	410	110	140	178
7	512	1390	4466	214	308	1214	124	200	358	112	182	250
8	366	1424	7988	150	420	1166	110	222	492	112	152	244
9	146	578	2292	162	224	662	134	138	290	138	148	212
10	330	1300	5156	126	340	942	128	234	426	112	184	218
11	298	1398	4664	188	492	1036	142	324	652	150	250	496
12	318	1764	6998	292	646	1334	226	424	856	206	350	608
13	614	2738	10872	270	680	2582	266	428	908	254	440	816
14	1326	4968	41684	504	1370	3174	576	1166	2148	378	834	1448
TOTAL =	4966	19332	100770	2536	5882	15304	2194	3992	7646	1946	3184	5254

Table 4.11 Number of RHS Evaluations Per Processor for the Best NWP Performance

from Table (4.8). The values of $RK(G_T)$ are obtained from the last row of Table (4.5) by selecting the smallest number from among the three that correspond to the selected G_T value. Thus we have

$$RK(10^{-3}) = 6506 \quad \text{(provided by RK7)}$$
$$RK(10^{-6}) = 15256 \quad \text{(provided by RK7)}$$
$$RK(10^{-9}) = 29609 \quad \text{(provided by RK9)}$$

The values of $SGA(G_T)$ are obtained from the last row of Table (4.6). The speed-up ratios S_1^{RK} and S_1^{SGA} are shown in Tables (4.12) and (4.13) respectively.

The second type of speed-up ratio which we consider is denoted by S_2^{RK} or S_2^{SGA}. Their values are shown in Tables (4.14) and (4.15) respectively. It is obtained as the average of the individual speed-up's taken over the fourteen test problems, rather than as a ratio of total performance as in the case of S_1. In particular

$$S_2^{RK}(k,G_T) = \frac{1}{14} \sum_{i=1}^{14} \frac{RK_i(G_T)}{NWP_i(k,G_T)}$$
$$S_2^{SGA}(k,G_T) = \frac{1}{14} \sum_{i=1}^{14} \frac{SGA_i(G_T)}{NWP_i(k,G_T)}.$$

$NWP_i(k,G_T)$ is the value in Table (4.11) for test problem TP_i that corresponds to the selected k and G_T values. The value of $RK_i(G_T)$ is obtain from Table (4.5) in the row corresponding to test

k \ GT	10^{-3}	10^{-6}	10^{-9}
k=2	1.31	0.79	0.29
k=4	2.57	2.59	1.93
k=6	2.97	3.82	3.87
k=8	3.34	4.79	5.64

Table 4.12 S_1^{RK} Speed-up Ratios

k \ GT	10^{-3}	10^{-6}	10^{-9}
k=2	0.80	0.42	0.14
k=4	1.57	1.40	0.93
k=6	1.82	2.06	1.87
k=8	2.05	2.58	2.72

Table 4.13 S_1^{SGA} Speed-up Ratios

k \ GT	10^{-3}	10^{-6}	10^{-9}
k=2	1.50	0.84	0.38
k=4	2.48	2.44	1.91
k=6	2.97	3.71	3.96
k=8	3.17	4.68	6.28

Table 4.14 S_2^{RK} Speed-up Ratios

k \ GT	10^{-3}	10^{-6}	10^{-9}
k=2	0.90	0.49	0.19
k=4	1.47	1.33	0.94
k=6	1.77	1.98	1.85
k=8	1.88	2.44	2.76

Table 4.15 S_2^{SGA} Speed-up Ratios

problem TP_i. When $G_T = 10^{-3}$, and 10^{-6}, the value from RK7 is used while RK9 is used for $G_T = 10^{-9}$. These choices are made because these methods provided the lowest total number of RHS evaluations at these accuracies and because they were also used in the calculation for the S_1 ratios. The value of $SGA_i(G_T)$ is taken from the row corresponding to test problem TP_i in Table (4.6) at the given G_T.

It is clear from Tables (4.12) and (4.14) that the S_1^{RK} and S_2^{RK} ratios are approximately the same and from Tables (4.13) and (4.15) that the S_1^{SGA} and S_2^{SGA} are also about the same. We also notice that the S_1^{SGA} and S_2^{SGA} are about half the values of S_1^{RK} and S_2^{RK}. Although the SGA code has a lower number of function evaluations relative to the Runge-Kutta methods, this code has a much higher overhead than both the Runge-Kutta methods and the BIPC methods.

Also, as a general observation, we note that the case k=2 does not show good performance. This is because of its low order relative to the methods with which it has been compared (7^{th} and 9^{th} order Runge-Kutta and variable order Adams). It should be noted furthermore that this inferior performance has been accentuated by not using the 5^{th} order Runge-Kutta results in the comparison. This choice arose because of our interest in comparing the BIPC methods with the most efficient of the Runge-Kutta codes being considered.

Perhaps the most important observation from the results described in this section is that the case k=8 gives the highest speed-up over all accuracies. Although relative to ther SGA code, the speed-up is not as high as desired it is, nevertheless, still significant particularly when the high overhead of this reference code is taken into account.

The discussion in this chapter has assumed a system realization in which the number of processors (N_p) is equal to the number of new solution values in each new block; i.e., k. Because of the direct coupling between k and the order of the integration process (ρ), the number of processors and ρ have likewise been implicitly coupled in our considerations. It is therefore of interest to briefly examine the desirability of attempting to decouple N_p and k (and hence N_p and ρ). The two simple possibilities are to have N_p as a multiple of k or k as a multiple of N_p. In the former case the number of processors exceeds the "natural parallelism" within the underlying procedure and hence all processors could be utilized only by segmenting the derivative vector. This alternative introduces considerations that are outside the scope of the study.

A specific, and entirely feasible, example of the second alternative is to implement a k=8 case using four processors. The question then arises as to whether or not such a utlization of four processors would yield better performance than a "conventional" k=4 implementation. Insight into this question can be

obtained from the speed-up ratios in Tables (4.12-4.15). We note first that such a "virtual" k=8 implementation would, at best, run at half the speed of an actual k=8 implementation. Thus this alternative is worthwhile only if the k=4 speed-up value is less than half the k=8 value for any particular G_T. This is indeed the case for the high accuracy solutions ($G_T=10^{-9}$) but is not the case for low accuracy solutions ($G_T=10^{-3}$). It is almost true for the moderate accuracy case of $G_T=10^{-6}$. A similar analysis can be undertaken for any other valid possibility but, on the basis of the data in Tables (4.12-4.15), the approach is most likely to be advantageous when higher solution accuracies are desired.

Chapter V

HARDWARE IMPLEMENTATION

5.1 INTRODUCTION

Experience with multiprocessor systems has shown that the minimization of interprocessor interference is one of the key design issues in exploiting parallelism. Making the right data available to the right processor at the right time is essential to keeping all the processors busy and this depends crucially upon the existence of the appropriate data paths. A number of articles give surveys on computer interconnection structures [Anderson & Jensen 1977; Siegel 1979]. Haynes et al [1982] summarize several practical interconnection networks using three parameters: cost, generality, and efficiency. Very efficient networks include crosspoint switches [Enslow 1977], reconfigurable buses [Arden & Giniosar 1982] and systolic arrays [Kung 1982]. The crosspoint and the reconfigurable buses represent very general but expensive interconnections.

Systolic arrays are examples of nearest neighbour processor connections, but very special examples - their communications and processors are optimized for specific problem classes. They provide extremely high speed and are very efficient, in that a large percentage of the hardware is in use at any given time. I/O

is overlapped with computation, but the crucial point is that each operand is input only once and is then operated upon many times. The problem of feeding the processors adequate amounts of data from the outside world is clearly vital. Therefore, the ratio of input operations to arithmetic operations is an important metric for evaluating their potential efficiency.

Some less efficient and less expensive interconnections ranging from the general purpose to specialized include the banyan [Goke & Lipovski 1973], k-cube [Locanthi 1980], Shuffle [Stone 1971] and tree [Despain & Patterson 1978]. The ring topology is a low cost, widely used general interconnection network; however it is relatively inefficient.

If a multiprocessor contains a shared memory for interprocessor communication, then a multiport memory may be used and elaborate interconnection networks avoided. However, multiport memories can give rise to implementation difficulties. The replicated memory concept is another approach for shared memory systems [Lillevik & Easterday 1983]. Such a memory structure consists of a set of memories, one for each processor, with identical contents. This eliminates read conflict since each processor can simply access its own private copy of the shared memory asynchronously or concurrently. To ensure shared memory integrity, write requests transfer data to all copies in parallel. However, because several processors may wish to write simultaneously, write requests require arbitration. The above system, therefore, eliminates read conflict but not write conflict.

However, for the class of BIPC integration methods discussed in Chapter 3, the processors terminate their computation at the same time and hence are required to exchange their data simultaneously and later read them concurrently. Thus, any implementation of this class of methods must eliminate both read and write conflicts.

The processor interconnection we are proposing is a very specialized one that is tailored to the class of BIPC methods. It is based on the idea of replicated shared memories with k memory elements for each processor where k is the number of processors. This approach makes it possible to eliminate both read and write conflicts, thus it satisfies the necessary requirements to correctly and efficiently support the class of BIPC methods.

5.2 FUNCTIONAL DESCRIPTION

Figure (5.1) illustrates, for a 4 processor case, the architecture of the proposed multi-microcomputer implementation of the class of BIPC methods. The number of processors used in the system will generally be equal to the block size of the method (N_p=k). As will be described below, the approach incorporates efficient methods of process synchronization and data transfer among the processors.

The user will typically specify the equations of the simulation model at the terminal of a suitable development system, e.g. MDS 225 [Intel 1983a]. To assist in program development,

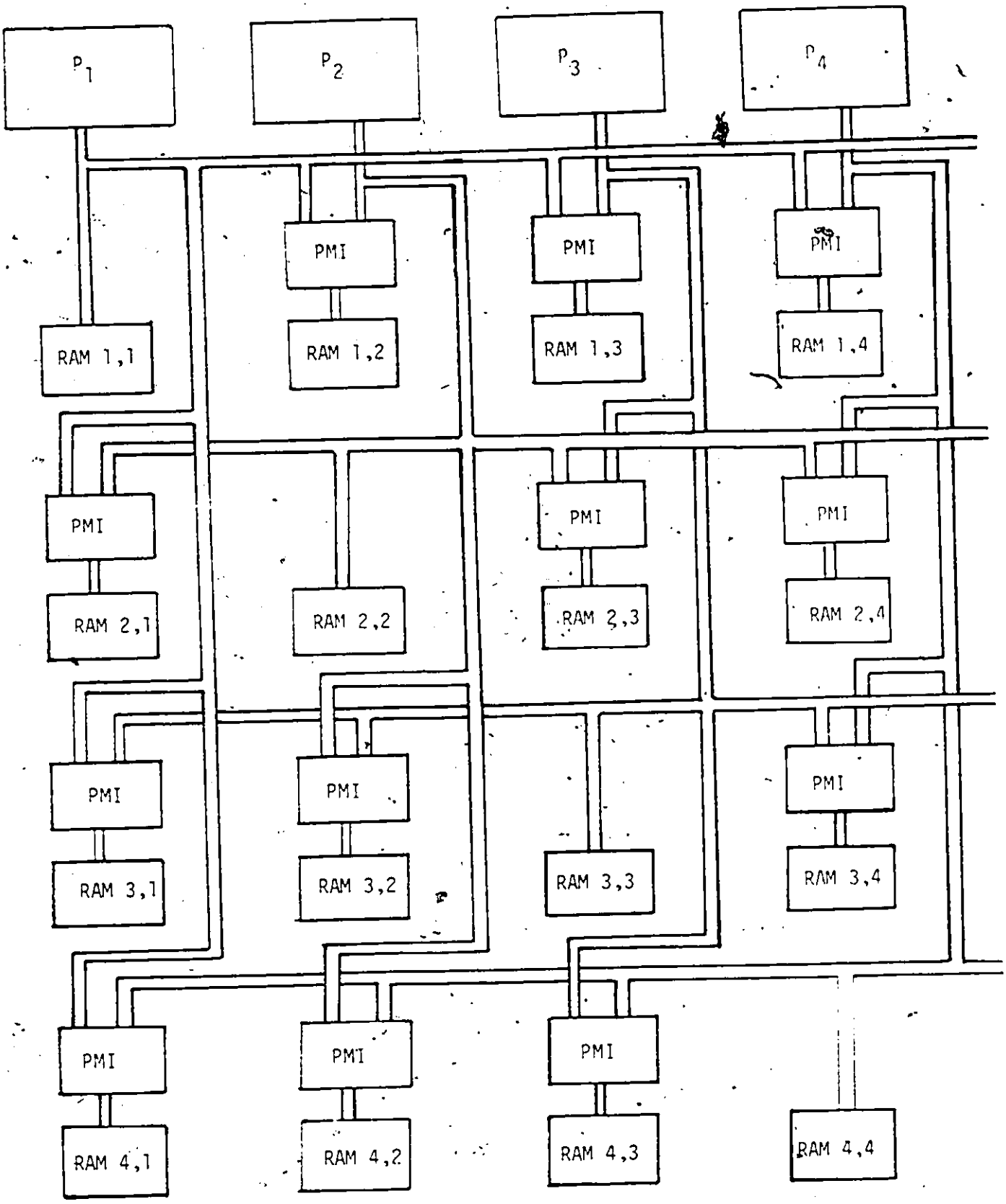

Figure 5.1: The Proposed Configuration (k=4)

the development system software would typically include an editor, a macro assembler, a high-level language compiler, a linking loader, and a library manager. With this support on the host system, the user prepares an object file which contains the system model for later down-loading to the microcomputers' local memories. Program execution, however, is controlled by the processors themselves.

Since the parallel tasks associated with the proposed integration algorithm are known in advance, the associated code is statically allocated to, and stored in the ROM of each processor. The code in processor i has the task of producing the solution value, y_i, together with its derivative, f_i. From an examination of the underlying formulas and the step adjustment procedures, it is clear that all processors have exactly the same computational load. This effectively eliminates synchronization problems.

The processors communicate their respective computed values through a k*k matrix of low cost memory elements. All processors write their computed values simultaneously with processor i writing its data in the memory element having subscript (i,i). However, the memory elements of a row are connected in parallel so that the data written in a memory element will be written simultaneously in all the memory elements of its row thereby creating identical information in the memory elements of each column. When reading data, each processor is given access to all k memory elements of its column. This replication of data in each column

provides the means for instantaneous data exchange among the processors and has very little overhead. The data flow direction for each of the off-diagonal memory elements is controlled by a processor memory interface (PMI) unit consisting of bidirectional bus buffers.

In Figure (5.2), the basic events during two cycles of the RIPC procedure are illustrated in the context of the four processor configuration. These include the data transfers to the memory array. Note that the elements in the k^{th} (in this case 4^{th}) row of the memory array are distinctive inasmuch as they hold four, rather than two, data values. We assume an 8-byte representation of the data and that all computation will be carried out in double precision using a numeric data processor. Since the memory elements of the 4^{th} row store 4 data values, a size of 2k bytes for each of these would be adequate for solving up to 64 ode's $(\frac{2048}{8x4})$. We regard this as a realistic number.

At step 1 (and 6), processor i generates the predicted value y_i^p and stores it in its local memory (y_i^p is required later in the error estimation process). The derivative values f_i^p is also generated and written to the memory element $RAM_{i,i}$. Because all memory elements of a row are connected in parallel, f_i^p will consequently be written in all memories of row i. At step 2 (and 7), processor i computes the solution at its point t_i, checks the error, calculates the corresponding R_i (see section 3.5.1) and writes it to the memory array. Since all R_i values are now avail-

	activity	Row 1		Row 2		Row 3		Row 4				
0	begin	f_{-3}	*	f_{-2}	*	f_{-1}	*	f_0	y_0	f_{-4}	*	
1	execute predictor step	f_{-3}	f_1^p	f_{-2}	f_2^p	f_{-1}	f_3^p	f_0	y_0	f_{-4}	f_4^p	
2	compute new solution values using corrector formulas, examine solution quality criterion and, if necessary reduce stepsize and return to step 1	f_{-3}	R_1	f_{-2}	R_2	f_{-1}	R_3	f_0	y_0	f_{-4}	R_4	
3	if solution quality acceptable, write solution values to memory array	f_{-3}	y_1	f_{-2}	y_2	f_{-1}	y_3	f_0	y_0	f_{-4}	y_4	
4	compute derivatives	f_{-3}	f_1	f_{-2}	f_2	f_{-1}	f_3	f_0	y_0	f_4	y_4	
5	rename (i → i-k)	*	f_{-3}	*	f_{-2}	*	f_{-1}	f_{-4}	*	f_0	y_0	(E1)
6	execute predictor step	f_1^p	f_{-3}	f_2^p	f_{-2}	f_3^p	f_{-1}	f_{-4}	f_4^p	f_0	y_0	
7	compute new solution values using corrector formulas, examine solution quality criterion and, if necessary reduce stepsize and return to step 1	R_1	f_{-3}	R_2	f_{-2}	R_3	f_{-1}	f_{-4}	R_4	f_0	y_0	
8	if solution quality acceptable, write solution values to memory array	y_1	f_{-3}	y_2	f_{-2}	y_3	f_{-1}	f_{-4}	y_4	f_0	y_0	
9	compute derivatives	f_1	f_{-3}	f_2	f_{-2}	f_3	f_{-1}	f_4	y_4	f_0	y_0	
10	rename (i → i-k)	f_{-3}	*	f_{-2}	*	f_{-1}	*	f_0	y_0	f_{-4}	*	(E2)

(E1) - end of first cycle, (E2) - end of second cycle, * - indicates available space

Figure 5.2 Event Sequence and Memory Array Updates
(Four Processor Configuration)

able to all processors, each processor then determines the largest of these values; i.e. R. If $R > 1$, the block computation is rejected and step 1 (or 6) is repeated with a reduced value of σ obtained from equation (3.24). Otherwise, the computation is considered successful and the solution values are written to the memory array. At step 4 (and 9), the derivatives at the new solution values are computed and written to the memory array. The remaining step corresponds to the index replacement $i \rightarrow (i-k)$ which simply reflects the labelling conventions between solution values within "old" and "new" blocks and has no relevance to the processing activity.

Since all processors write their data at the same time after evaluation of essentially identical formulas, a simple and efficient synchronization mechanism can be used. A processor, having reached a synchronization point in the program, executes an output instruction and waits until all other processors reach the same synchronization point (this will necessarily occur almost simultaneously). One bit of an output port of each processor is AND-connected for this purpose as shown in Figure (5.3). When S is one, it enables all processors to proceed with computation until the next synchronization point.

It is significant to note that the proposed architecture can also provide for fault tolerance because of the replication of data in the memory array. Once a processor is detected to be

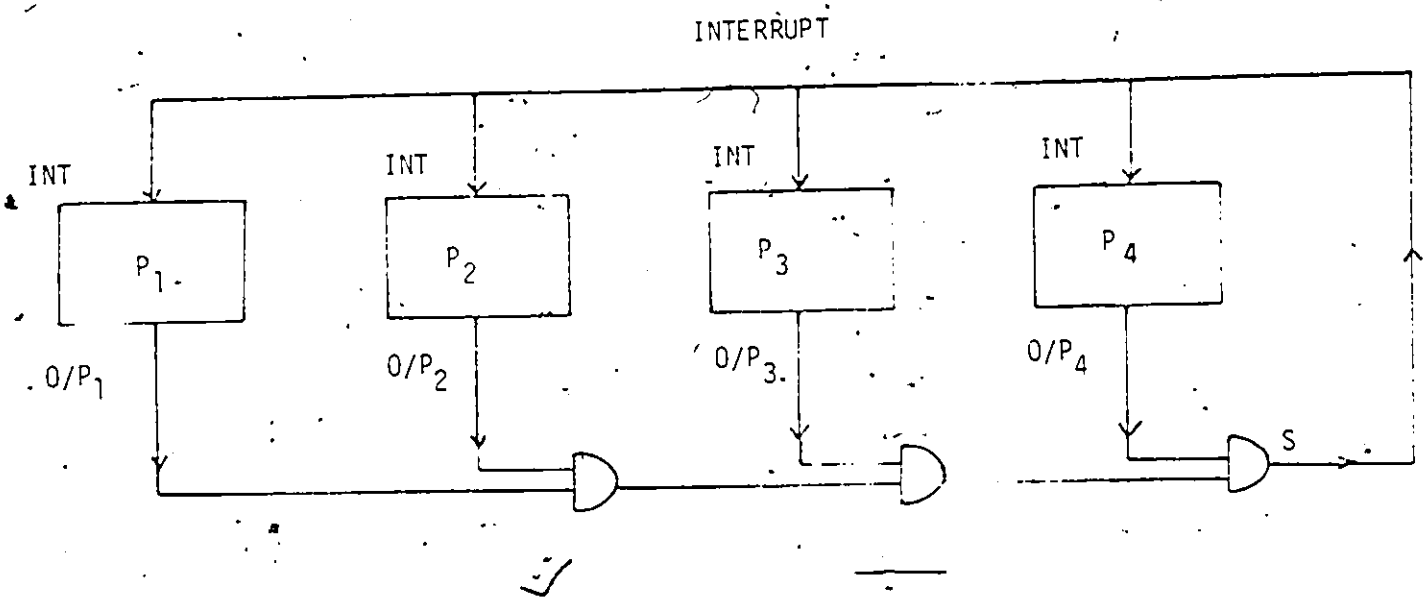

Figure 5.3: Synchronization Mechanism

faulty, it is eliminated together with its memory elements (the whole column is removed) and the remaining processors would run a lower blocksize formula (smaller k) stored in their local ROM's. Similarly, in the case of a memory failure, the faulty memory as well as all other memory elements in its column are eliminated together with their corresponding processor. Thus, the architecture has the added desirable feature of providing for fail soft tolerance.

In the following section we suggest a system implementation based on the Intel product line and specifically on the iSBC 86/05 single board microcomputer. The board is based on the 16-bit iAPX 86/10 microprocessor. It is assumed that the board will include an 8097 Numeric Data Processor which provides a hardware implementation of double precision arithmetic. This is a highly desirable feature in our intended application. The design could easily be modified to accomodate other hardware components.

5.3 SYSTEM IMPLEMENTATION

In the preceding section, a functional description of the system was presented. This section explains how these functions are implemented.

5.3.1 Address Bus Buffering

When interfacing to microprocessor-controlled systems, one

must constantly keep in mind the electrical "load" being placed on any digital signal line. By load we mean the amount of electric current a particular digital signal line is required to supply. Each input of a device presents a load on the output driving it. Buses cannot feed an unlimited number of circuits. Because of this buffers or drivers are used in large systems to boost the driving power of the buses [Bibbero & Stern 1982; Aumieux 1982; Zaks & Lesea 1979].

As shown in Figure (5.4), address lines (ADR0/-ADRA/) from the Multibus of each processor [Intel 1981b] are buffered and inverted using 74LS04 inverters. LS-Series logic (Low-power Schottky-diode-clamped) is used for buffering because its I_{IL} value of -0.4 mA is consistent with the Multibus constraint of -0.8mA maximum [Intel 1979a]. S-Series (Schottky-diode-clamped) or standard series logic will not meet this specification since these have I_{IL} equal to -2 mA and -1.6 mA respectively [Texas 1971]. I_{IH}, however, is satisfied for all series. The buffered address lines of each processor are then sent to all the memory elements on its row and column to select a specific location.

5.3.2 Bus Address Decoding

This logic decodes the appropriate Multibus address bits of each processor into RAM requests. The address decoding circuit diagram is shown in Figure (5.4).

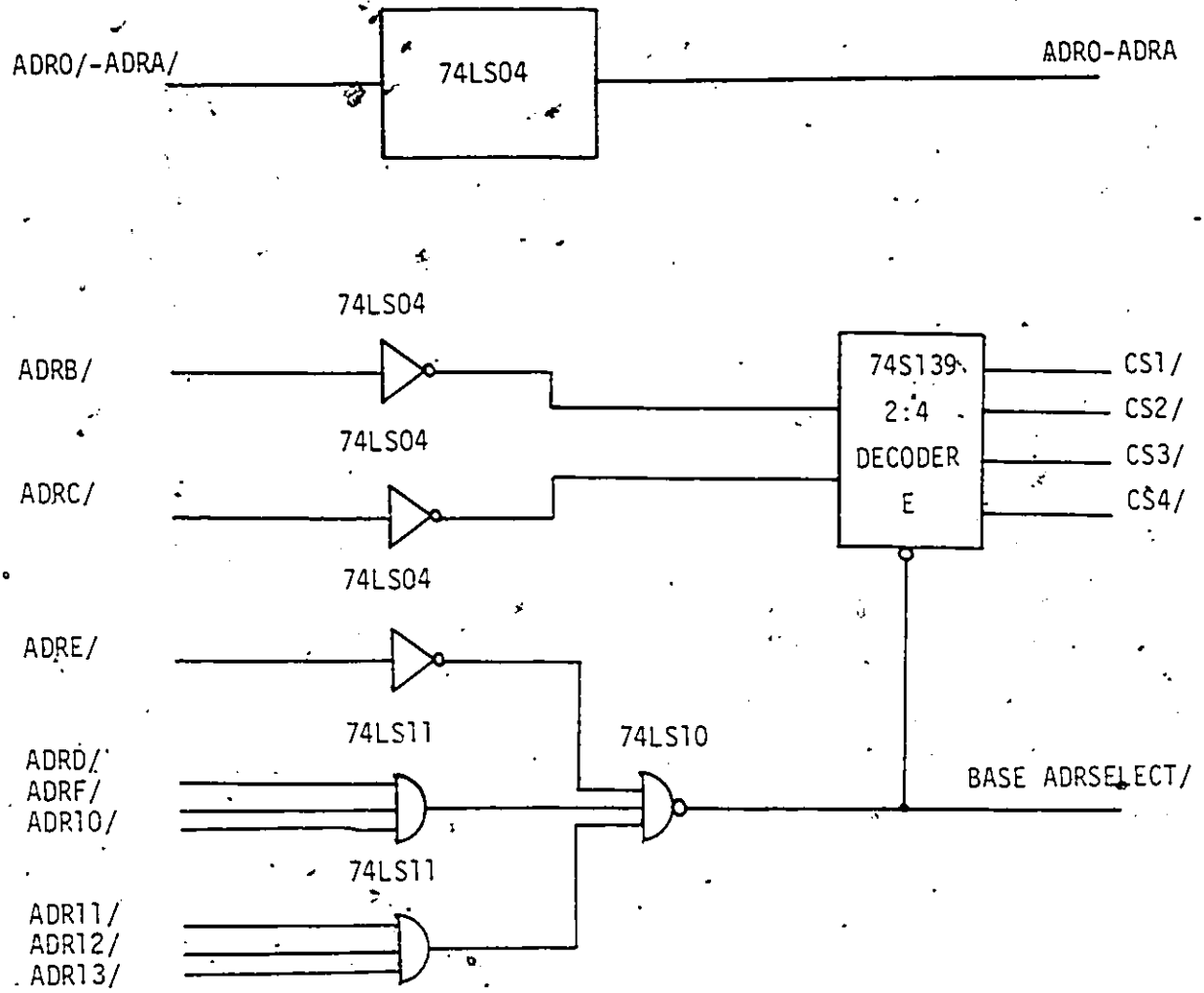

Figure 5.4: Address Decoding Circuit

Within the context of the four processor system being discussed, each processor can access any of the four memory elements in its column of 2K-bytes each. Address lines (ADRD/, ADRF/-ADR13/) are AND-ed together using 74LS11 AND gates. LS-Series logic is required to meet the Multibus specification. Address line (ADRE/) is inverted by a 74LS04 inverter and AND-ed with address lines (ADRD/, ADRF/-ADR13/) by a 74S10 NAND gate to form the BASE ADRSELECT/ signal. This signal specifies the base address of an 8K-byte memory having the address 04000 to 05FFF. It enables a 74S139 two-line-to-four-line decoder for decoding chip selects for the row memories. Address lines (ADRB/-ADRC/) are buffered and inverted using a 74LS04 inverter and then input to the decoder so that the 4 row memories will have the addresses 04000 to 047FF, 04800 to 04FFF, 05000 to 057FF, and 05800 to 05FFF respectively.

The maximum propagation delays for the 74LS11, 74LS04, 74S10, and 74S139 are 15 ns, 15 ns, 5 ns and 10 ns respectively [Texas 1976]. Hence the chip select signals will be delayed by a maximum of 15+5+10=30 ns from the application of stable address lines and will preceed memory read/write commands (MRDC/,MWTC/) since these are applied at least 50 ns after the application of stable address. These delays will be important when considering memory read and write timings.

5.3.3 Control Signal Logic

The control signal logic consists of the circuits that

forward the memory read/write commands from the Multibus of each processor to their respective destinations, provide the bus with a transfer acknowledge response, and drive the system interrupt lines.

a) Control Signals

Three Multibus control signals are used in the proposed system; namely

(i) the Memory Read Command (MRD/) which indicates that the address of a memory location is on the Multibus interface address lines and that the contents of that location are to be placed on the Multibus interface data lines,

(ii) the Memory Write Command (MWTC/) which indicates that the address of a memory location is on the Multibus interface address lines and that the contents on the Multibus interface data lines are to be written into that location, and

(iii) the Transfer Acknowledge (XACK/) that indicates that the addressed memory location has completed the specified read or write operation.

Because the bus DC requirements specify that each board may load the MRDC/ and MWTC/ lines with 2.0 mA, Schottky devices are used [Intel 1979a]. The circuit diagram for generating the control signals is shown in Figure (5.5). The MRDC/ and MWTC/ are qualified by the BASE ADRSELECT/ signal to form the RD and WT signals. The RD/ and WT/ signals will be delayed by a maximum of 10.5 ns from the application of MRDC/ and MWTC/.

b) **Transfer Acknowledge Generation**

The user interface transfer acknowledge generation logic provides a transfer acknowledge response, XACK/, to notify the bus master that write data provided by the bus master has been accepted or that read data it has requested is available on the Multibus data lines. XACK/ allows the bus master to conclude its current instruction. The transfer acknowledge signal must be driven by three state drivers to meet the Multibus DC requirements. The driver is enabled when the bus interface is addressed and a command is present.

The circuit which generates the transfer acknowledge is shown in Figure (5.5). The RD and WT signals are OR-ed to form the BD ENBL/ signal which is inverted and used to drive the CLEAR pin of a 74164 shift register. When the slave board is not being accessed, the CLEAR pin of the shift register will be low (BD ENBL/ is high). This causes the shift register to remain cleared and all outputs of the shift register will be low. When the slave

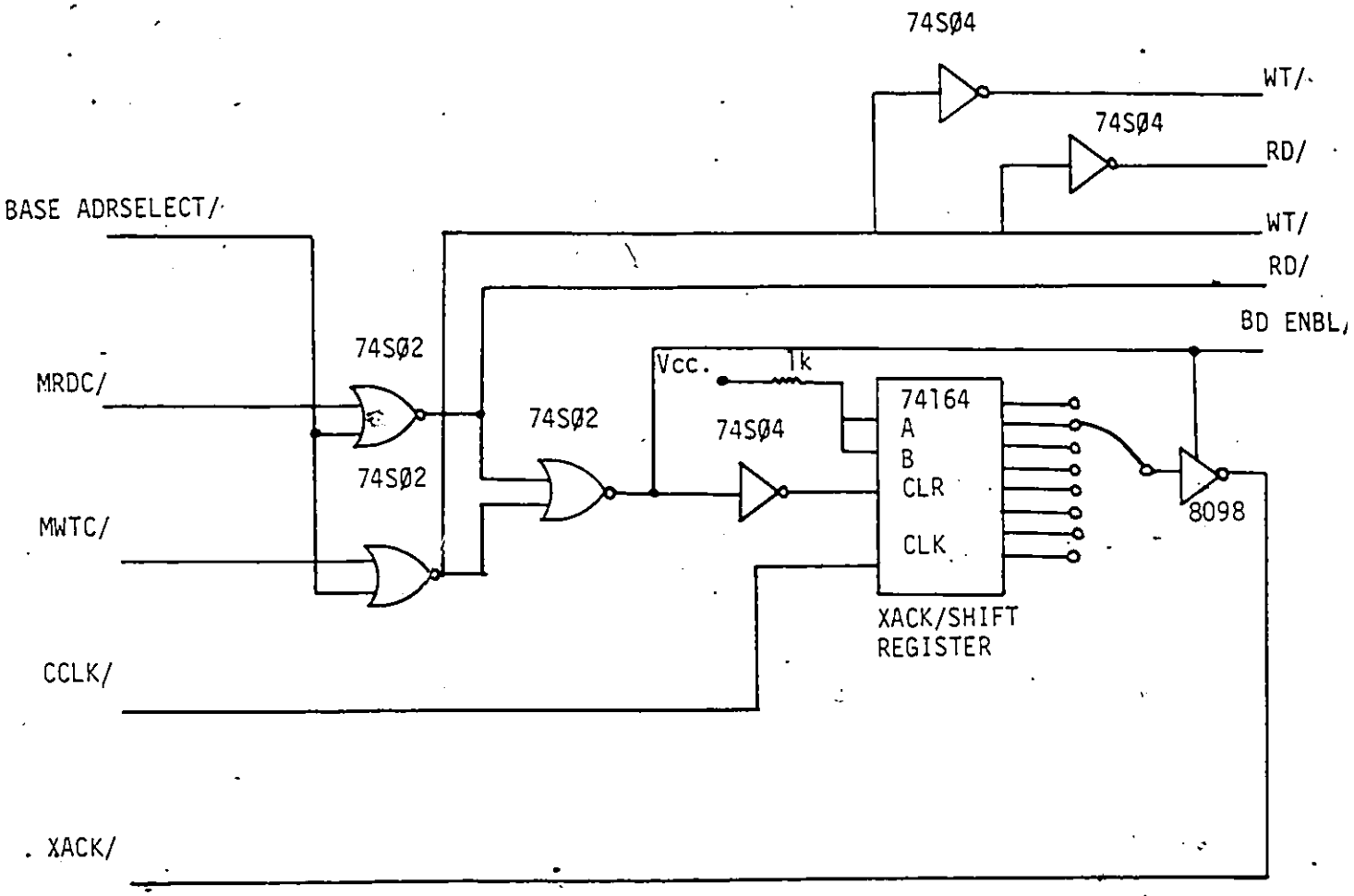

Figure 5.5 Control Signals and Transfer Acknowledge

board is accessed, the CLEAR pin will be high, and the A and B inputs (which are high) will be clocked to the output pins by a Constant Clock (CCLK/) provided by the Multibus. To select a delay for the XACK/ signal, a jumper must be installed from one of the shift register output pins to the 8089 tri-state driver. Each of the shift register output pins selects an integer multiple of CCLK/ periods for the signal delay. Since the CCLK/ signal is asynchronous, the actual delay selected may only be specified with a tolerance of one CCLK/ period. In this design, a delay of 1-2 CCLK/ periods is selected; with a CCLK/ period of 100 ns, the XACK/ delay would occur somewhere within the range of 100-200 ns from the time when the CLEAR signal goes high. This delay is selected because it is well suited to the access time of the memory element used in this design.

c) Interrupt Logic

The type of interrupt used in this design is a Non Bus Vectored interrupt in which the interrupt vector address is generated by the bus master. This does not require the Multibus address lines for transfer of the interrupt vector address. The interrupt vector address is generated by the interrupt controller on the master and transferred to the processor over the local bus. The source of the interrupt can be on the master module or on other bus modules, in which case the bus modules use the Multibus interrupt request lines (INT0/-INT7) to generate their interrupt requests to the bus master. When an interrupt request line is

activated, the bus master performs its own interrupt operation and processes the interrupt. The asynchronous interrupt lines must be driven by open collector devices with a minimum drive of 16 mA [Intel 1979a].

In the proposed system, since all processors write their data at their designated area at the same time, a processor, having reached a synchronization point in a program, executes an output instruction and waits until all other processors reach the same synchronization point. Off-board peripheral operations are handled in the iSBC 86/05 through three, 8-bit, parallel I/O ports having the addresses CA, CC, and C8 and are connected to external equipment via edge connector J1 [Intel 1981a]. Using port C8, each processor sends a byte equal to 1 which will set bit 0 of port C8 (J1-47). These bits are AND-ed together and output through an open collector gate to the Multibus interrupt lines (INT1/) of each processor. The interrupt logic circuit diagram is shown in Figure (5.6). The interrupt request is cleared by performing an output of a byte equal to 0 by all processors to port C8 at the end of their Interrupt Service Routines.

5.3.4 Data Bus Buffers

In the general case, the communication media consists of a k*k matrix of low cost random access memory elements where k is the number of processor. Processor i can write data only in $RAM_{i,i}$. However, all memory elements of a row are connected in

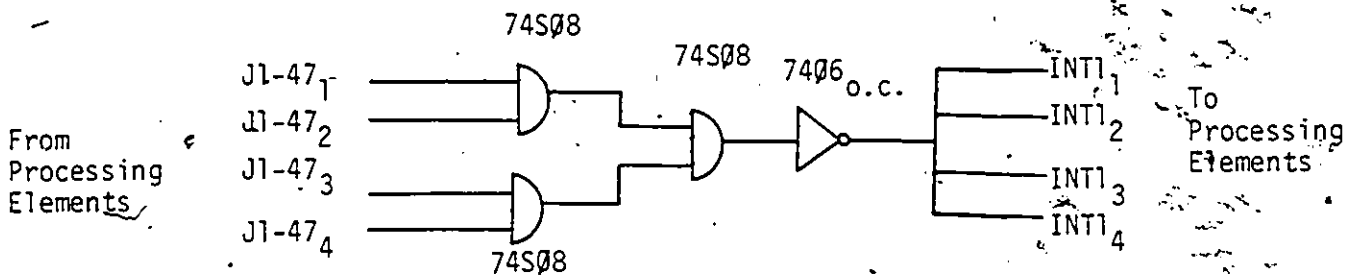

Figure 5.6: Interrupt Logic

parallel and this leads to the simultaneous transfer of data among all row memories. All processors can however read data independently, each from their respective column memories.

A processor-memory interface (PMI) must be provided for each memory element (except the diagonal memories) in order to control the direction of the data transfer. This is shown in Figure (5.7). Processor i can write into $RAM_{i,j}$ while processor j can only read from it. Four octal bus transceivers, each consisting of an Intel 8287 are used. The two that are connected to the data lines of processor i ($DATO/_i$-$DATF/_i$) are always in the transmit mode (T is high) while the two others, connected to the data lines of processor j ($DATO/_j$- $DATA/_j$) are in the receive mode (T is low). However, the output (as selected by T) is not enabled unless the output enable signal ($\overline{OE}$) is low. Transceivers A and C are enabled when processor i is writing ($WT/_i$ is low) in row i ($CSi/_i$ is low) while transceivers B and D are enabled when processor j is reading ($RD/_j$ is low) from $RAM_{i,j}$ ($CSi/_j$ is low).

5.3.5 Memory Elements and Data Transfer

There are two basic types of circuit technology for semiconductor RAMs: static and dynamic. While dynamic RAM is denser and less expensive, and has lower power dissipation than static RAM, it requires more complex drive circuitry and has more critical timing requirements.

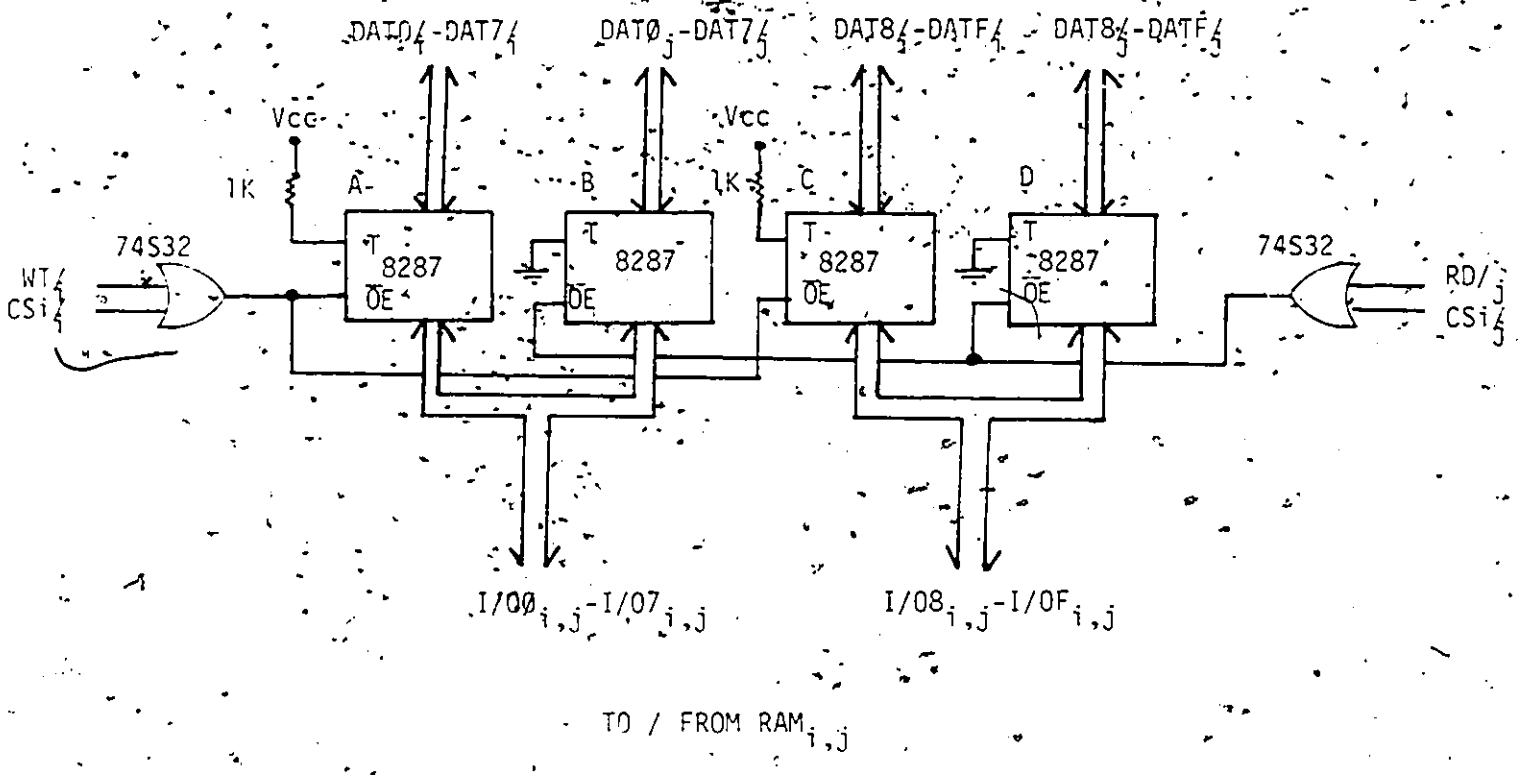

Figure 5.7 : Processors-Memory Interface (PMI)

Each memory element used in this design consists of four 2148H-3 [Intel 1983b] common I/O Static Random Access Memories arranged in four columns. The 2148H-3 is a 4096-bit organized as 1K words by 4-bits so that the size of each memory element will be 1K words by 16-bits. The 2148H-3 has a minimum read and write cycle times of 55 ns which is compatible with the Multibus MRDC/ and MWTC/ pulse width of 100 ns. The address, data, and control signals from processors i and j to $RAM_{i,j}$ are shown in Figure (5.8).

$RAM_{i,j}$ will be in the write mode only if processor i is writing ($\overline{WT}/_i$ is connected to the memory $\overline{WE}$). Otherwise, it will be in the read mode. The buffered address lines (ADR1-ADRA) of both processors i and j are connected to the address lines of $RAM_{i,j}$ through 74LS158 2-line-to-1-line multiplexers. Address lines ($ADR1_i$ - $ADRA_i$) are selected only if processor i is writing (WT_i is high), otherwise address lines ($ADR1_j$ - $ADRA_j$) are selected.

It is interesting to note the reason for the use of the LS-Series for the multiplexers rather than the S-Series. For the four processor case, each buffered address line must drive one input line for each of six multiplexers. The number of devices of the same electrical type a particular device is able to drive is known as the "fan out" of the device [Coffron & Long 1983]. This

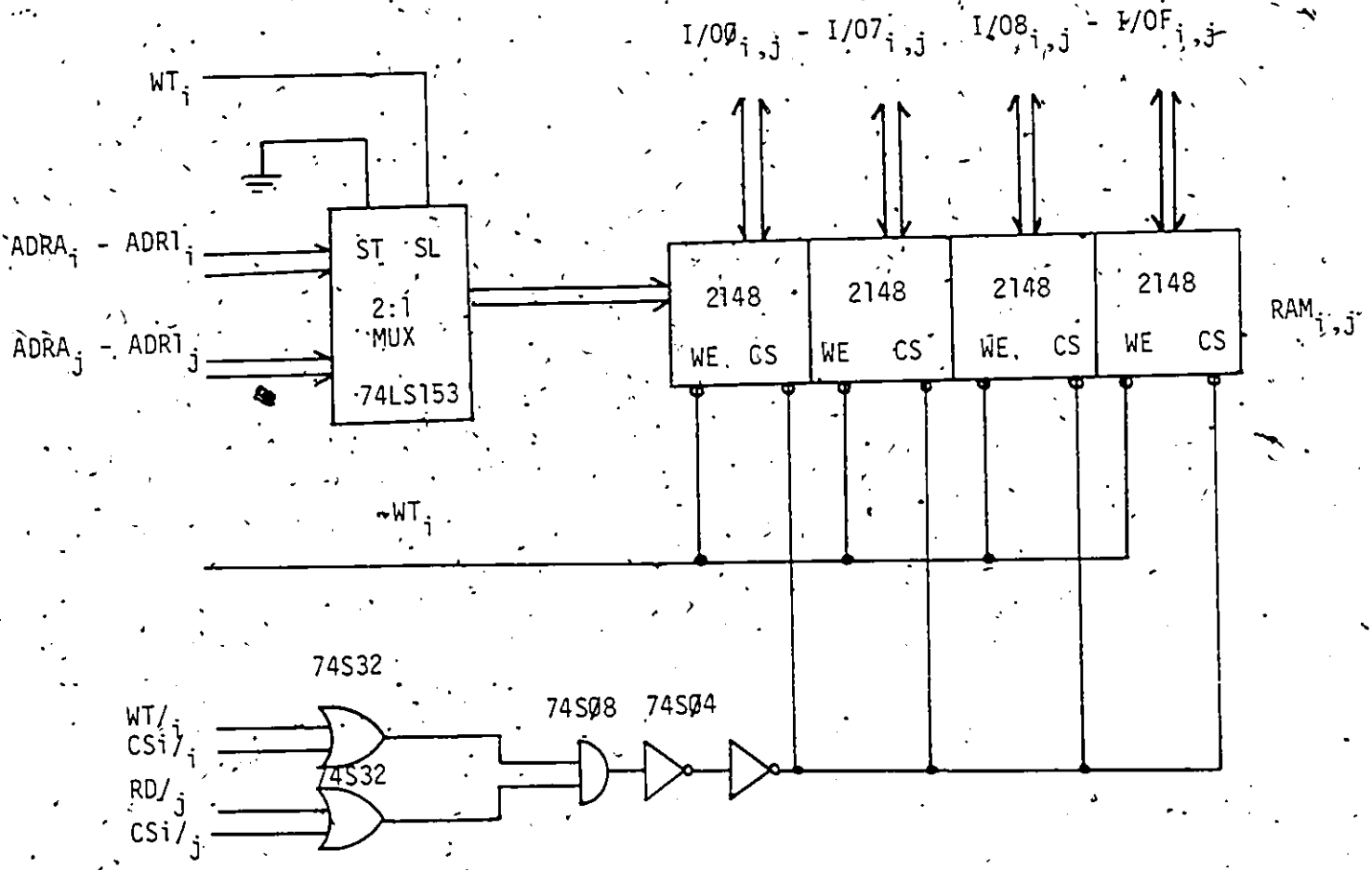

Figure 5.8: Memory Address, Data, and Control Signals

is equal to $\min \left(\frac{I_{OL}}{I_{IL}}, \frac{I_{OH}}{I_{IH}}\right)$ where I_{OL} and I_{OH} are the logical 0 and logical 1 output currents of the driving device and I_{IL} and I_{IH} are the sum of logical 0 and logical 1 input currents respectively of the driven devices. These electrical parameters can be obtained from the data sheets of the different devices. With the use of the LS-Series,

$$\text{Fan out} = \min \left(\frac{400}{20}, \frac{8}{0.4}\right) = 20$$

which means that each address line buffered by a 74LS04 can drive up to twenty input lines of a 74LS158. However if the S-Series is used then the fan out is only equal to four which is less than the number of lines required to be driven. Therefore, LS-Series should be used for the multiplexers.

A price is paid by having a larger propagation delay for the 74LS158 than that of the 74S158. The 74LS158 has a maximum propagation delay of 24 ns, and since the WT signal occurs after 5.5 ns from the MWTC/ command, then the address to the memory element becomes valid after at most 29.5 ns from the MWTC/ command. The 2148H-3 specifications imply that addresses should be valid prior to or coincide with $\overline{CS}$ transition low (t_{AS} = 0 ns MIN). $RAM_{i,j}$ is selected only if processor j is reading from it ($RD/_j$ and $CSi/_j$ are both low) or processor i is writing in the i^{th} row ($WT/_i$ and $CSi/_i$ are both low). These signals are AND-ed to form the chip select for the memory element. Since RD/ and WT/ signals

arrive after at most 10.5 ns from the application of MRDC/ or MWTC/ and since the maximum delay is 7.5 ns for 74S08 and 7 ns for 74S32, then the memory element is selected after at most 25 ns from the application of MRDC/ or MWTC/. This means that $\overline{CS}$ is valid prior to address lines become valid which violates the 2148H-3 specifications. Two 74S04 inverters are inserted, as shown in Figure (5.8), to delay the $\overline{CS}$ by 9.5 ns so that it becomes valid after the appearance of stable addresses.

This section has given a detailed design approach for the proposed multi-micrcomputer system based on the Intel product line. However, the approach could equally be undertaken using hardware components from other manufacturers.

Chapter VI

SUMMARY AND CONCLUSIONS

The research reported in this thesis has explored a possible means for exploiting multiple processors in the solution of continuous system simulation problems. This has been undertaken in terms of

a) an investigation of a class of parallel variable stepsize integration methods that are based on block implicit predictor-corrector formulas,

b) the design of a computer architecture with a special communication mechanism well suited to the proposed class of parallel methods.

The underlying goal in such a multiprocessor system application is, of course, the achievement of faster solutions of the ode's embedded within continuous system simulation problems. This issue has been explored via an extensive set of numerical experiments.

Parallel variable stepsize integration methods have not been previously reported in the literature. The class of such methods investigated in this research project are intended for use with

continuous system models that are characterized as being "non-stiff". It is not likely that these methods will provide good peformance when applied to stiff problems.

Two particular subclasses of methods have been evaluated which we refer to as the FWP and NWP forms. Stability analysis, complexity considerations and the experimental results obtained, all suggest the superiority of the latter approach which is newly proposed.

It is important to note that the investigated integration methods are parallel in nature which makes it natural to statically assign the different tasks required by these methods to separate processors, and makes task allocation completely problem independent. This overcomes the problems associated with the equations segementation approach where the user or a compiler has the difficult task of trying to evenly balance the workload over the available processors. The synchronous nature of these methods greatly simplifies task scheduling and task synchronization.

A multiprocessor architecture for realizing the proposed parallel integration algorithm has also been proposed where the processors are interconnected through a replicated shared memory. The memory structure consists of a two dimensional array of low cost memory elements. Such an interconnection structure eliminates both read and write conflicts and allows the data to be instantaneously exchanged among all processors. This feature

makes the proposed structure well suited to the synchronous nature of these integration methods since all the processors terminate their computation and become ready to exchange their data at the same instant in time.

The considerable decrease in interprocessor interference (the elimination of both read and write conflicts) provided by such an interconnection structure definitely increases the multiprocessor throughput. In addition, it may provide for increased fault tolerance because of the replication of data in the proposed structure.

Although, a detailed design of the proposed architecture is presented based on the 8086 microprocessor, the architecture is a general-purpose infrastraction; i.e., it can be easily upgraded to more powerful microprocessors (e.g. 818086, 828086,..).

Because a hardware realization of the proposed system is not yet available, the reported performance results have been obtained from simulation studies in a single processor environment. Our results obtained in this framework do establish the practicality of the approach as an effective means of achieving parallelism, and hence speed-up, in the solution of the ode's within continuous system simulation problems. However it must be emphasized that actual performance gains for any specific problem will likely differ from those that are reported in this study. There are two major reasons for this; namely,

a) The results are based on a particular limited set of test problems.

b) The results are based on an imperfect solution time measure (count of derivative function evaluations) which ignores some portion of the workload carried out by the various integration methods compared in the study

The extent to which performance variations occur over the applications domain will certainly require further experimentation. With regard to (b), we note that its influence does tend to vanish as the complexity of the derivative functions increases since then the relative contribution of the ignored workload is diminished.

The highest value of k (the number of new solution values generated in each new block) considered in the experimental investigations described in Chapter 4, is eight. It is natural, therefore, to wonder if a deterioration in performance should be anticipated as k is increased beyond this value. From an examination of the absolute stability boundary results given in Table (3.1), it is clear that for values of k beyond fourteen, the absolute stability boundary significantly decreases. From this it could be inferred that a maximum practical value for k would probably occur at k=14.

Our consideration have principally focused on the case where

$k=N_p$ (here N_p is the number of processors used in the implementation). Two alternate situations; namely the case where k is a multiple of N_p and the case where N_p is a multiple of k, are briefly discussed at the end of Chapter 4.

The proposed NWP form of the BIPC approach has been demonstrated in our study to be an effective means for achieving parallelism in the simulation studies of continuous systems. Further experimentation with the approach based on an actual hardware implementation of the suggested architecture would be a fruitful avenue for research. Further research into other parallel integration algorithms for the solution of ode's would also be worthwhile, particularly in the context of special problem classes; e.g., stiff systems.

Appendix A

DEMONSTRATION OF THE NON-SINGULARITY OF THE MATRIX U

In this appendix we demonstrate the non-singularity of the kxk matrix

$$U = [u^1\ u^2\ \ldots\ldots\ u^k]$$

where $u = \begin{bmatrix} 1 \\ 2 \\ \vdots \\ k \end{bmatrix}$ and $u^j = \begin{bmatrix} 1^j \\ 2^j \\ \vdots \\ k^j \end{bmatrix}$

Recall first that the Vandermonde matrix

$$\Lambda = \begin{bmatrix} 1 & 1 & \ldots & 1 \\ \lambda_1 & \lambda_2 & & \lambda_k \\ \vdots & & & \\ \lambda_1^{k-1} & \lambda_2^{k-1} & \ldots & \lambda_k^{k-1} \end{bmatrix}$$

is non singular provided $\lambda_i \neq \lambda_j$. If we set $\lambda_j = j$, then

$$\Lambda^* = \begin{bmatrix} 1 & 1 & \ldots & 1 \\ 1 & 2 & & k \\ 1 & 2^2 & & k^2 \\ \vdots & \vdots & & \vdots \\ 1 & 2^{k-1} & & k^{k-1} \end{bmatrix}$$

is non-singular

Let D_u be the kxk diagonal matrix whose diagonal entries are 1, 2, 3, ... k. Then $Q = \Lambda^* D_u$ is non-singular and furthermore $Q = U^T$, hence U also is non-singular.

Appendix B

CORRECTOR AND PREDICTOR FORMULAS

$$y_1 = y_0 + \frac{h}{12}(5f_0 + 8f_1 - f_2) + \frac{h^4 y[4]}{24}$$

$$y_2 = y_0 + \frac{h}{3}(f_0 + 4f_1 + f_2) - \frac{h^5 y[5]}{90}$$

Corrector Formulas for k=2

$$y_1 = y_0 + \frac{h}{720}(251f_0 + 646f_1 - 264f_2 + 106f_3 - 19f_4) + \frac{3h^6 y[6]}{160}$$

$$y_2 = y_0 + \frac{h}{90}(29f_0 + 124f_1 + 24f_2 + 4f_3 - f_4) + \frac{h^6 y[6]}{90}$$

$$y_3 = y_0 + \frac{3h}{80}(9f_0 + 34f_1 + 24f_2 + 14f_3 - f_4) + \frac{3h^6 y[6]}{160}$$

$$y_4 = y_0 + \frac{2h}{45}(7f_0 + 32f_1 + 12f_2 + 32f_3 + 7f_4) - \frac{8h^7 y[7]}{945}$$

Corrector Formulas for k=4

$$y_1 = y_0 + \frac{h}{60480}(19087f_0 + 65112f_1 - 46461f_2 + 37504f_3 - 20211f_4 + 6312f_5 - 863f_6) + \frac{275h^8y[8]}{24192},$$

$$y_2 = y_0 + \frac{h}{3780}(1139f_0 + 5640f_1 + 33f_2 + 1328f_3 - 807f_4 + 264f_5 - 37f_6) + \frac{8h^8y[8]}{945};$$

$$y_3 = y_0 + \frac{h}{2240}(685f_0 + 3240f_1 + 1161f_2 + 2176f_3 - 729f_4 + 216f_5 - 29f_6) + \frac{9h^8y[8]}{896},$$

$$y_4 = y_0 + \frac{2h}{945}(143f_0 + 696f_1 + 192f_2 + 752f_3 + 87f_4 + 24f_5 - 4f_6) + \frac{8h^8y[8]}{945},$$

$$y_5 = y_0 + \frac{5h}{12096}(743f_0 + 3480f_1 + 1275f_2 + 3200f_3 + 2325f_4 + 1128f_5 - 55f_6) + \frac{275h^8y[8]}{24192},$$

$$y_6 = y_0 + \frac{h}{140}(41f_0 + 216f_1 + 27f_2 + 272f_3 + 27f_4 + 216f_5 - 41f_6) - \frac{9h^9y[9]}{1400},$$

Corrector Formulas for k=6

$$y_1 = y_0 + \frac{h}{3628800}\,(1070017f_0 + 4467094f_1 - 4604594f_2 + 5595358f_3 - 5033120f_4 + 3146338f_5 - 1291214f_6 + 312874f_7 - 33953f_8) + \frac{8183h^{10}\,y[10]}{1036800},$$

$$y_2 = y_0 + \frac{h}{113400}\,(32377f_0 + 182584f_1 - 42494f_2 + 120088f_3 - 116120f_4 + 74728f_5 - 31154f_6 + 7624f_7 - 833f_8) + \frac{9h^{10}\,y[10]}{1400},$$

$$y_3 = y_0 + \frac{h}{44800}\,(12881f_0 + 70902f_1 + 3438f_2 + 79934f_3 - 56160f_4 + 34434f_5 - 14062f_6 + 3402f_7 - 369f_8) + \frac{25h^{10}\,y[10]}{3584},$$

$$y_4 = y_0 + \frac{h}{14175}\,(4063f_0 + 22576f_1 + 244f_2 + 32752f_3 - 9080f_4 + 9232f_5 - 3956f_6 + 976f_7 - 107f_8) + \frac{94h^{10}\,y[10]}{14175},$$

$$y_5 = y_0 + \frac{5h}{145152}\,(8341f_0 + 46030f_1 + 1510f_2 + 63670f_3 - 800f_4 + 34186f_5 - 9830f_6 + 2290f_7 - 245f_8) + \frac{25h^{10}\,y[10]}{3584},$$

$$y_6 = y_0 + \frac{h}{1400}\,(401f_0 + 2232f_1 + 18f_2 + 3224f_3 - 360f_4 + 2664f_5 + 158f_6 + 72f_7 - 9f_8) + \frac{9h^{10}\,y[10]}{1400},$$

$$y_7 = y_0 + \frac{7h}{518400}\,(21361f_0 + 116662f_1 + 6958f_2 + 155134f_3 + 7840f_4 + 105154f_5 + 74578f_6 + 31882f_7 - 1169f_8) + \frac{8183h^{10}\,y[10]}{1036800}$$

$$y_8 = y_0 + \frac{4h}{14175}\,(989f_0 + 5888f_1 - 928f_2 + 10496f_3 - 4540f_4 + 10496f_5 - 928f_6 + 5888f_7 + 989f_8) - \frac{2368h^{11}\,y^{[11]}}{467775}$$

Corrector Formulas for k=8

$$y_1 = \frac{1}{3}\sum_{j=0}^{2} y_{-j} + \frac{h}{6}(13f_0 - 4f_{-1} + 3f_{-2}) + \frac{13}{36}\,h^4\,y^{[4]},$$

$$y_2 = \frac{1}{3}\sum_{j=0}^{2} y_{-j} + \frac{h}{12}(79f_0 - 72f_{-1} + 29f_{-2}) + \frac{191}{72}\,h^4\,y^{[4]}.$$

Predictor Formulas for k=2 (EWP Form)

$$y_1 = \frac{1}{5}\sum_{j=0}^{4} y_{-j} + \frac{h}{240}(697f_0 - 686f_{-1} + 936f_{-2} - 322f_{-3} + 95f_{-4}) + \frac{461}{1440}\,h^6\,y^{[6]},$$

$$y_2 = \frac{1}{5}\sum_{j=0}^{4} y_{-j} + \frac{h}{360}(4411f_0 - 9226f_{-1} + 10272f_{-2} - 5110f_{-3} + 1093f_{-4}) + \frac{2369}{720}\,h^6\,y^{[6]},$$

$$y_3 = \frac{1}{5}\sum_{j=0}^{4} y_{-j} + \frac{h}{720}(26083f_0 - 68026f_{-1} + 78600f_{-2} - 41974f_{-3} + 8917f_{-4}) + \frac{22477}{1440}\,h^6\,y^{[6]},$$

$$y_4 = \frac{1}{5}\sum_{j=0}^{4} y_{-j} + \frac{h}{120}(10469f_0 - 30430f_{-1} + 36576f_{-2} - 20258f_{-3} + 4363f_{-4}) + \frac{37369}{720}\,h^6\,y^{[6]}.$$

Predictor Formulas for k=4 (EWP Form)

$$y_1 = \frac{1}{7}\sum_{j=0}^{6} y_{-j} + \frac{h}{15120}\,(53623f_0 - 93528f_{-1} + 178299f_{-2} - 157376f_{-3}$$
$$+ 102789f_{-4} - 28584f_{-5} + 50257f_{-6}) + \frac{599}{2016}\,h^8\,y^{[8]},$$

$$y_2 = \frac{1}{7}\sum_{j=0}^{6} y_{-j} + \frac{h}{60480}\,(1158251f_0 - 2841704f_{-1} + 6980175f_{-2} - 7177600f_{-3}$$
$$+ 4449825f_{-4} - 1486296f_{-5} + 219749f_{-6}) + \frac{156727}{40320}\,h^8\,y^{[8]},$$

$$y_3 = \frac{1}{7}\sum_{j=0}^{6} y_{-j} + \frac{h}{5040}\,(358081f_0 - 1449472f_{-1} + 2788637f_{-2} - 3014208f_{-3}$$
$$+ 1908067f_{-4} - 657824f_{-5} + 96959f_{-6}) + \frac{139597}{6048}\,h^8\,y^{[8]},$$

$$y_4 = \frac{1}{7}\sum_{j=0}^{6} y_{-j} + \frac{h}{60480}\,(12716059f_0 - 56823336f_{-1} + 114325983f_{-2}$$
$$- 127578752f_{-3} + 82402353f_{-4} - 28921176f_{-5} + 4302229f_{-6})$$
$$+ \frac{3797047}{40320}\,h^8\,y^{[8]},$$

$$y_5 = \frac{1}{7}\sum_{j=0}^{6} y_{-j} + \frac{h}{15120}\,(8054999f_0 - 38190456f_{-1} + 79396446f_{-2} - 90685312f_{-3}$$
$$59543973f_{-4} - 21179016f_{-5} + 3180329f_{-6}) + \frac{3069043}{10080}\,h^8\,y^{[8]},$$

$$y_6 = \frac{1}{7}\sum_{j=0}^{6} y_{-j} + \frac{h}{20160}\,(24269689f_0 - 119694904f_{-1} + 255020357f_{-2}$$
$$- 296535164f_{-3} + 19732122f_{-4} - 70941512f_{-5} + 10741751f_{-6})$$
$$+ \frac{20254753}{24192}\,h^8\,y^{[8]}.$$

Predictor Formulas for k=6 (EWP Form)

$$y_1 = \frac{1}{9}\sum_{j=0}^{8} y_{-j} + \frac{h}{725760}\,(3004583f_0 - 7622438f_{-1} + 18962082f_{-2}$$
$$- 26660782f_{-3} + 27128800f_{-4} - 17395218f_{-5} + 7709918f_{-6}$$
$$- 1729562f_{-7} + 231417f_{-8}) + \frac{245167}{870912}\,h^{10}\,y[10],$$

$$y_2 = \frac{1}{9}\sum_{j=0}^{8} y_{-j} + \frac{h}{907200}\,(24893233f_0 - 112534074f_{-1} + 284780974f_{-2}$$
$$- 442897138f_{-3} + 455181120f_{-4} - 308077838f_{-5} + 134096594f_{-6}$$
$$- 33613254f_{-7} + 3813983f_{-8}) + \frac{24240229}{5443200}\,h^{10}\,y[10],$$

$$y_3 = \frac{1}{9}\sum_{j=0}^{8} y_{-j} + \frac{h}{3628800}\,(440498979f_0 - 2420823422f_{-1} + 6535840682f_{-2}$$
$$10639010214f_{-3} + 11227891360f_{-4} - 7769475994f_{-5}$$
$$+ 3425581782f_{-6} - 874105922f_{-7} + 99004349f_{-8})$$
$$\frac{690292759}{21772800}\,h^{10}\,y[10];$$

$$y_4 = \frac{1}{9}\sum_{j=0}^{8} y_{-j} + \frac{h}{907200}\,(386336569f_0 - 2331561082f_{-1} + 6593351742f_{-2}$$
$$- 11072331314f_{-3} + 11937052160f_{-4} - 8396317134f_{-5}$$
$$+ 3747016642f_{-6} - 966472582f_{-7} + 110182599f_{-8})$$
$$+ \frac{833495221}{5443200}\,h^{10}\,y[10],$$

$$y_5 = \frac{1}{9}\sum_{j=0}^{8} y_{-j} + \frac{h}{3628800}\,(4583551043f_0 - 29263180734f_{-1} + 85530264874f_{-2}$$
$$- 146979767398f_{-3} + 161143175520f_{-4} - 114829956698f_{-5}$$
$$+ 51770784854f_{-6} - 13467789954f_{-7} + 1545577693f_{-8})$$
$$+ \frac{12606752791}{21772800}\,h^{10}\,y[10],$$

$$y_6 = \frac{1}{9}\sum_{j=0}^{8} y_{-j} + \frac{h}{181440}\,(599522877f_0 - 3974084722f_{-1} + 11902451158f_{-2} -$$
$$20819930058f_{-3} + 23135614400f_{-4} - 16662821942f_{-5}$$
$$+ 7577212842f_{-6} - 1985339278f_{-7} + 229189123f_{-8})$$
$$+ \frac{57298219}{31104}\,h^{10}\,y[10],$$

$$y_7 = \frac{1}{9}\sum_{j=0}^{8} y_{-j} + \frac{h}{3628800}\,(28434102307f_0 - 193611570046f_{-1} + 590810335146f_{-2}$$
$$- 1048099707302f_{-3} + 1177556024480f_{-4} - 855663112602f_{-5}$$
$$+ 391953895126f_{-6} - 103330739266f_{-7} + 11990688957f_{-8})$$
$$+ \frac{112052192791}{3110400}\,h^{10}\,y[10],$$

$$y_8 = \frac{1}{9}\sum_{j=0}^{8} y_{-j} + \frac{h}{907200}\,(15574257401f_0 - 108205367418f_{-1} + 335093847358f_{-2}$$
$$- 601288802866f_{-3} + 681762147840f_{-4} - 499129908686f_{-5}$$
$$+ 230075288258f_{-6} - 60979158918f_{-7} + 7108583431f_{-8})$$
$$+ \frac{70664375221}{5443200}\,h^{10}\,y[10].$$

Predictor Formulas for k=8 (EWP Form)

$$y_1 = y_0 + \frac{h}{12}(23f_0 - 16f_{-1} + 5f_{-2}) + \frac{3}{8}h^4 y^{[4]},$$

$$y_2 = y_0 + \frac{h}{3}(19f_0 - 20f_{-1} + 7f_{-2}) + \frac{8}{3}h^4 y^{[4]}$$

Predictor Formulas for k=2 (NWP Form)

$$y_1 = y_0 + \frac{h}{720}(1901f_0 - 2774f_{-1} + 2616f_{-2} - 1274f_{-3} + 251f_{-4}) + \frac{95}{288}h^6 y^{[6]},$$

$$y_2 = y_0 + \frac{h}{90}(1079f_0 - 2396f_{-1} + 2544f_{-2} - 1316f_{-3} + 269f_{-4}) + \frac{33}{10}h^6 y^{[6]},$$

$$y_3 = y_0 + \frac{3h}{80}(959f_0 - 2546f_{-1} + 2904f_{-2} - 1566f_{-3} + 329f_{-4}) + \frac{2499}{160}h^6 y^{[6]},$$

$$y_4 = y_0 + \frac{2h}{45}(1957f_0 - 5728f_{-1} + 6852f_{-2} - 3808f_{-3} + 817f_{-4}) + \frac{2336}{45}h^6 y^{[6]}.$$

Predictor Formulas for k=4 (NWP Form)

$$y_1 = y_0 + \frac{h}{60480}\,(198721f_0 - 447288f_{-1} + 705549f_{-2} - 688256f_{-3} + 407139f_{-4} - 134472f_{-5} + 19087f_{-6}) + \frac{5257}{17280}\,h^8\,y^{[8]},$$

$$y_2 = y_0 + \frac{h}{3780}\,(71405f_0 - 244680f_{-1} + 43583f_{-2} - 452272f_{-3} + 277863f_{-4} - 94152f_{-5} + 13613f_{-6}) + \frac{736}{189}\,h^8\,y^{[8]},$$

$$y_3 = y_0 + \frac{h}{2240}\,(158563f_0 - 646920f_{-1} + 1239111f_{-2} - 1341824f_{-3} + 847881f_{-4} - 293112f_{-5} + 43021f_{-6}) + \frac{103437}{4480}\,h^8\,y^{[8]},$$

$$y_4 = y_0 + \frac{2h}{945}\,(99221f_0 - 444504f_{-1} + 893112f_{-2} - 997168f_{-3} + 643737f_{-4} - 226104f_{-5} + 33596f_{-6}) + \frac{17800}{189}\,h^8\,y^{[8]},$$

$$y_5 = y_0 + \frac{5h}{12096}\,(1288169f_0 - 6113400f_{-1} + 12703125f_{-2} - 14512000f_{-3} + 9526875f_{-4} - 3389448f_{-5} + 508775f_{-6}) + \frac{7365875}{24192}\,h^8\,y^{[8]},$$

$$y_6 = y_0 + \frac{h}{140}\,(168503f_0 - 831384f_{-1} + 1770957f_{-2} - 2059408f_{-3} + 1370277f_{-4} - 492696f_{-5} + 74591f_{-6}) + \frac{29304}{35}\,h^8\,y^{[8]}.$$

Predictor Formulas for k=6 (NWP Form)

$$y_1 = y_0 + \frac{h}{3628800}\,(14097247f_0 - 43125206f_{-1} + 95476786f_{-2} - 139855262f_{-3} + 137968480f_{-4} - 91172642f_{-5} + 38833486f_{-6} - 9664106f_{-7} + 1070017f_{-8}) + \frac{25713}{89600}\,h^{10}\,y[10],$$

$$y_2 = y_0 + \frac{h}{113400}\,(3057727f_0 - 14223416f_{-1} + 35618446f_{-2} - 55566872f_{-3} + 56970280f_{-4} - 38640872f_{-5} + 16770946f_{-6} - 4233416f_{-7} + 473977f_{-8}) + \frac{20225}{4536}\,h^{10}\,y[10],$$

$$y_3 = y_0 + \frac{h}{44800}\,(5426831f_0 - 29948598f_{-1} + 80697618f_{-2} - 131426686f_{-3} + 138644640f_{-4} - 95971266f_{-5} + 42294638f_{-6} - 10803978f_{-7} + 1221201f_{-8}) + \frac{719179}{22680}\,h^{10}\,y[10],$$

$$y_4 = y_0 + \frac{h}{14175}\,(6032893f_0 - 36450224f_{-1} + 103023724f_{-2} - 173030768f_{-3} + 186525520f_{-4} - 131208848f_{-5} + 58548244f_{-6} - 15105104f_{-7} + 1721263f_{-8}) + \frac{26798}{175}\,h^{10}\,y[10],$$

$$y_5 = y_0 + \frac{5h}{145152}\,(36661003f_0 - 234145550f_{-1} + 684247450f_{-2} - 1175890550f_{-3} + 1289164000f_{-4} - 918673226f_{-5} + 414168550f_{-6} - 107750450f_{-7} + 12363925f_{-8}) + \frac{168091625}{290304}\,h^{10}\,y[10],$$

$$y_6 = y_0 + \frac{h}{1400}\,(4625591f_0 - 30666168f_{-1} + 91840158f_{-2} - 160650136f_{-3} + 178516440f_{-4} - 128572776f_{-5} + 58466258f_{-6} - 15319368f_{-7} + 1768401f_{-8}) + \frac{368431}{6400}\,h^{10}\,y[10],$$

$$y_7 = y_0 + \frac{7h}{518400}\,(580268911f_0 - 3951358838f_{-1} + 12057367378f_{-2} - 21389923646f_{-3} + 24031803040f_{-4} - 17462598146f_{-5} + 7999064878f_{-6} - 2108811338f_{-7} + 244706161f_{-8}) + \frac{2634975}{512}\,h^{10}\,y[10],$$

$$y_8 = y_0 + \frac{4h}{14175}(60836039f_0 - 422682112f_{-1} + 1308960992f_{-2} - 2348790784f_{-3}$$
$$+ 2663135660f_{-4} - 1949730304f_{-5} + 898731872f_{-6}$$
$$- 238200832f_{-7} + 27767819_{-8}) \frac{184021888}{14175} h^{10} y[10].$$

Predictor Formulas for k=8 (NWP Form)

Appendix C

VARIABLE STEPSIZE PREDICTOR FORMULAS (EWP FORM)

$$y_i^p = \frac{1}{3} \sum_{j=0}^{2} y_{-j} + h * \sum_{m=0}^{2} f_{-m} * (V_m (i\sigma) + \theta_m) + C_i^p h^4 y^{[4]} \qquad i = 1,2$$

where:

$$V_0(s) = \frac{1}{12}(2s^3 + 9s^2 + 12s) \quad ; \theta_0 = \frac{1}{4}$$

$$V_1(s) = \frac{-1}{3}(s^3 + 3s^2) \quad ; \theta_1 = \frac{2}{3}$$

$$V_2(s) = \frac{1}{12}(2s^3 + 3s^2) \quad ; \theta_2 = \frac{1}{12}$$

$$C_i^p = \frac{1}{4!}(s^4 + 4s^3 + 4s^2 - \frac{1}{3}) \quad ; \; i = 1,2 \quad s = i\sigma$$

Variable Stepsize Predictor Formulas For k=2 (EWP Form)

$$y_i^p = \frac{1}{5} \sum_{j=0}^{4} y_{-j} + h * \sum_{m=0}^{4} f_{-m} * (V_m(i\sigma) + \theta_m) + C_i^p h^6 y^{[6]}$$

$$i = 1,2,3,4$$

where:

$$V_0(s) = \frac{1}{720}(6s^5 + 75s^4 + 350s^3 + 750s^2 + 720s) \quad ; \theta_{72} = \frac{19}{72}$$

$$V_1(s) = \frac{-1}{360}(12s^5 + 135s^4 + 520s^3 + 720s^2) \quad ; \theta_1 = \frac{179}{180}$$

$$V_2(s) = \frac{1}{60}(3s^5 + 30s^4 + 95s^3 + 90s^2) \quad ; \theta_2 = \frac{4}{15}$$

$$V_3(s) = \frac{-1}{360}(12s^5 + 105s^4 + 280s^3 + 240s^2) \quad ; \theta_3 = \frac{77}{180}$$

$$V_4(s) = \frac{1}{720}(6s^5 + 45s^4 + 110s^3 + 90s^2) \quad ; \theta_4 = \frac{17}{360}$$

$$C_i^p = \frac{1}{6!}(s^6 + 12s^5 + 52.5s^4 + 100s^3 + 72s^2 - 7) \quad ; i=1,..,4$$

$$s = i\sigma$$

Variable stepsize Predictor Formulas For k=4 (EWP Form)

$$y_i^p = \frac{I}{7} \sum_{j=0}^{6} y_{-j} + h * \sum_{m=0}^{6} f_{-m} * (V_m(i\sigma) + \Theta_m) + C_i^p h^8 y^{[8]}$$

$$i = 1,2,\ldots,6$$

where:

$$V_0(s) = \frac{1}{60480}(12s^7 + 294s^6 + 2940s^5 + 15435s^4 + 45472s^3 + 74088s^2 + 60480s) \quad ; \Theta_0 = \frac{751}{2880}$$

$$V_1(s) = \frac{-1}{2520}(3s^7 + 70s^6 + 651s^5 + 3045s^4 + 7308s^3 + 7560s^2) \quad ; \Theta_1 = \frac{3049}{2520}$$

$$V_2(s) = \frac{1}{20160}(60s^7 + 1330s^6 + 11508s^5 + 48405s^4 + 98280s^3 + 75600s^2) \quad ; \Theta_2 = \frac{2549}{20160}$$

$$V_3(s) = \frac{-1}{3780}(15s^7 + 315s^6 + 2541s^5 + 9765s^4 + 17780s^3 + 12600s^2 \quad ; \Theta_3 = \frac{34}{35}$$

$$V_4(s) = \frac{1}{20160}(60s^7 + 1190s^6 + 8988s^5 + 32235s^4 + 55440s^3 + 37800s^2) \quad ; \Theta_4 = \frac{1339}{20160}$$

$$V_5(s) = \frac{-1}{2520}(3s^7 + 56s^6 + 399s^5 + 1365s^4 + 2268s^3 + 1512s^2) \quad ; \Theta_5 = \frac{839}{2520}$$

$$V_6(s) = \frac{1}{60480}(12s^7 + 210s^6 + 1428s^5 + 4725s^4 + 7672s^3 + 5040s^2) \quad ; \Theta_6 = \frac{647}{20160}$$

$$C_i^p = \frac{1}{8!}(s^8 + 24s^7 + \frac{700}{3}s^6 + 1176s^5 + 3248s^4 + 4704s^3 + 2880s^2 - \frac{859}{3}) \quad ; i = 1,2,..,6$$

$$s = i\sigma$$

Variable Stepsize Predictor Formulas For k=6 (EWP Form)

$$y_i^p = \frac{1}{g} \sum_{j=0}^{8} y_{-j} + h * \sum_{m=0}^{8} f_{-m} * (V_m(i\sigma) + \Theta_m) + C_i^p h^{10} y[10]$$

$$i = 1,2,\ldots,8$$

where:

$$V_0(s) = \frac{1}{3628800}(10s^9 + 405s^8 + 7020s^7 + 68040s^6 + 404082s^5 + 1513890s^4 + 3543720s^3 + 4931280s^2 + 3628800s) \quad ; \Theta_0 = \frac{2857}{11208}$$

$$V_1(s) = \frac{-1}{1814400}(40s^9 + 1575s^8 + 26280s^7 + 241500s^6 + 1326528s^5 + 4397400s^4 + 8311680s^3 + 7257600s^2) \quad ; \Theta_1 = \frac{626627}{453600}$$

$$V_2(s) = \frac{1}{1814400}(140s^9 + 5355s^8 + 86040s^7 + 751800s^6 + 3852828s^5 + 11562390s^4 + 18779040s^3 + 12700800s^2) \quad ; \Theta_2 = \frac{-83297}{453600}$$

$$V_3(s) = \frac{-1}{1814400}(280s^9 + 10395s^8 + 160920s^7 + 1341900s^6 + 6483456s^5 + 18075960s^4 + 26920320s^3 + 16934400s^2) \quad ; \Theta_3 = \frac{90991}{50400}$$

$$V_4(s) = \frac{1}{181440}(35s^9 + 1260s^8 + 18810s^7 + 150360s^6 + 692559s^5 + 1835820s^4 + 2611980s^3 + 1587600s^2) \quad ; \Theta_4 = \frac{-1816}{2835}$$

$$V_5(s) = \frac{-1}{1814400}(280s^9 + 9765s^8 + 140760s^7 + 1084020s^6 + 4810176s^5 + 12325320s^4 + 17055360s^3 + 10160640s^2 \quad ; \Theta_5 = \frac{524569}{453600}$$

$$V_6(s) = \frac{1}{1814400}(140s^9 + 4725s^8 + 65880s^7 + 491400s^6 + 2119068s^5 + 5301450s^4 + 7200480s^3 + 4233600s^2) \quad ; \Theta_6 = \frac{3943}{50400}$$

$$V_7(s) = \frac{-1}{1814400}(40s^9 + 1305s^8 + 17640s^7 + 128100s^6 + 540288s^5 + 1328040s^4 + 1779840s^3 + 1036800s^2) \quad ; \Theta_7 = \frac{127037}{453600}$$

$$V_8(s) = \frac{1}{3628800}\,(10s^9 + 315s^8 + 4140s^7 + 29400s^6 + 121842s^5 + 295470s^4 + 392040s^3 + 226800s^2) \qquad ; \; o_8 = \frac{21767}{907200}$$

$$\bar{C}_i^p = \frac{1}{10!}\,(s^{10} + 40s^9 + 682.5s^8 + 6480s^7 + 37415s^6 + 134568s^5 + 295310s^4 + 365280s^3 + 201600s^2 - \frac{59542}{3}) \qquad i = 1,2,\ldots,8$$

$$s = i\sigma$$

Variable Stepsize Predictor Formulas For k=8 (EWP Form)

Appendix D

SPECIFICATIONS OF THE TEST PRORLEMS

TP1: $\dot{y} = -y$; $y(0) = 1$

; $t_f = 20$

Analytic Solution

$y = e^{-t}$

TP2: $\dot{y} = \frac{y^3}{12}$; $y(0) = 1$

; $t_f = 20$

Analytic Solution

$y = \frac{1}{\sqrt{t+1}}$

TP3: $\dot{y} = y \cos t$; $y(0) = 1$

; $t_f = 20$

Analytic Solution

$y = e^{\sin t}$

TP4: $\dot{y} = \frac{y}{4}\left(1 - \frac{y}{20}\right)$; $y(0) = 1$

; $t_f = 20$

Analytic Solution

$y = \frac{20}{1 + 19\, e^{-t/4}}$

TP5: $\dot{y}_1 = -y_2 - y_1 y_3/r$; $y_1(0) = 3$

$\dot{y}_2 = y_1 - y_2 y_3/r$; $y_2(0) = 0$

$\dot{y}_3 = y_1/r$; $y_3(0) = 0$

with $r = (y_1^2 + y_2^2)^{\frac{1}{2}}$; $t_f = 20$

Analytic Solution

$y_1 = (2 + \cos(t)) \cos(t)$

$y_2 = (2 + \cos(t)) \sin(t)$

$y_3 = \sin(t)$

TP6: $\dot{y}_1 = y_2$; $y_1(0) = 1$

$\dot{y}_2 = -y_1/r^3$; $y_2(0) = 0$

$\dot{y}_3 = y_4$; $y_3(0) = 0$

$\dot{y}_4 = -y_3/r^3$; $y_4(0) = 1$

with $r = (y_1^2 + y_3^2)^{\frac{1}{2}}$; $t_f = 25$

Analytic Solution

$y_1 = \cos(t)$

$y_2 = -\sin(t)$

$y_3 = \sin(t)$

$y_4 = \cos(t)$

TP7: $\dot{y}_1 = y_2$; $y_1(0) = 1$

$\dot{y}_2 = -2y_1^2 (1 - 4t^2 y_1)$; $y_2(0) = 0$

; $t_f = 20$

Analytic Solution

$$y_1 = \frac{1}{1+ t^2}$$

$$y_2 = \frac{-2t}{(1+t^2)^2}$$

TP8: $\dot{y}_1 = \frac{y_1}{2(1+t)} - 2t\, y_2$; $y_1(0) = 1$

$\dot{y}_2 = \frac{y_2}{2(1+t)} + 2t\, y_1$; $y_2(0) = 0$

; $t_f = 6$

Analytic Solution

$$y_1 = (1+t)^{\frac{1}{2}} \cos(t^2)$$

$$y_2 = (1+t)^{\frac{1}{2}} \sin(t^2)$$

TP9: $\dot{y}_1 = y_2$; $y_1(0) = 0$

$\dot{y}_2 = -2y_2 - 101y_1$; $y_2(0) = 1$

$\dot{y}_3 = y_4$; $y_3(0) = 0$

$\dot{y}_4 = y_1 - 4y_4 - 29y_3$; $y_4(0) = 0$

; $t_f = 5$

Analytic Solution

$$y_1 = 0.1e^{-t}\sin(10t)$$

$$y_2 = (1.01)^{\frac{1}{2}}e^{-t}\cos(10t+\theta_1)$$

$$y_3 = (5e^{-t}\sin(10t-\psi)+10e^{-2t}\sin(5t+\phi))/Q$$

$$y_4 = (5e^{-t}(101)^{\frac{1}{2}}\cos(10t-(\psi-\theta_1)+10e^{-2t}(29)^{\frac{1}{2}}\cos(5t-(\phi-\theta_2)))/Q$$

with $Q = 50\ (6476)^{\frac{1}{2}}$

$\psi = \tan^{-1}(-20/74)$

$\phi = \tan^{-1}(-10/76)$

$\theta_1 = \tan^{-1}(0.1)$

$\theta_2 = \tan^{-1}(0.4)$

TP10: $\dot{y}_1 = y_3$; $y_1(0) = 1-\varepsilon$

$\dot{y}_2 = y_4$; $y_2(0) = 0$

$\dot{y}_3 = -y_1/(y_1^2 + y_2^2)^{3/2}$; $y_3(0) = 0$

$\dot{y}_4 = -y_2/(y_1^2 + y_2^2)^{3/2}$; $y_4(0) = \left(\frac{1+\varepsilon}{1-\varepsilon}\right)^{\frac{1}{2}}$

$\varepsilon = 0.1$

TP11: As above with $\varepsilon = 0.3$

TP12: As above with $\varepsilon = 0.5$

TP13: As above with $\varepsilon = 0.7$

TP14: As above with $\varepsilon = 0.9$

Analytic Solution

$y_1 = \cos(u) - \varepsilon$

$y_2 = (1-\varepsilon^2)^{\frac{1}{2}} \sin(u)$

$y_3 = \dfrac{-\sin u}{1 - \varepsilon \cos(u)}$

$y_4 = \dfrac{(1-\varepsilon^2)^{\frac{1}{2}}\cos(u)}{1- \varepsilon \cos(u)}$

where $u - \varepsilon \sin(u) - t = 0$

BIBLIOGRAPHY

Abou-Rabia, O., Birta, L.G., 1984, "A Multi-microprocessor System for Continuous System Simulation," Proceeding, 1984 Summer Computer Simulation Conference, Boston, MA, pp. 27-33.

Abou-Rabia, O., Birta, L.G., 1985, "The Design of a Multi-microcomputer System for Continuous System Simulation," submitted for publication.

Adkins, G., Pooch, V.W., 1977, "Computer Simulation: A Tutorial," Computer, Vol. 10, No. 4, pp. 12-19.

Amyot, J.R., 1983, "Computer Studies of AVC Heave Performance as a Function of Vent Valve Propotional Control Parameters", Canadian Aeronautics and Space Journal, Vol. 29, No. 1, pp. 25-34.

Anderson, G.A., Jensen, E.D., 1977, "Computer Interconnection Structures: Taxonomy, Characteristics, and Examples," Computing Surveys, Vol. 7, No. 4, pp. 197-213.

Andriessen, J.H.M., et al., 1984, "The Delft Parallel Processor/A Distributed MIMD Processor," Proceeding, 1984 Summer Computer Simulation Conference, Boston, MA., pp. 1241-1246.

Arden, B.W., Ginosar, R., 1982, "MP/C: A Multiprocessor/Computer Architecture," IEEE Trans. Computers, Vol. C-31, No. 5, pp. 455-473.

Aumieux, M., 1982, Microprocessor Systems, John Wiley and Sons, N.Y.

Backus, J., 1978, "Can Programming be Liberated from the Von Neumann Style? A Functional Style and its Algebra of Programs," Communications of the ACM, Vol. 21, No. 8, pp. 613-641.

Baker, C.T.H., Phillips, C., 1981, The Numerical Solution of Nonlinear Problems, Clarendon Press, Oxford.

Ribbero, R.J., Stern, D.M., 1982, Microprocessor Systems - Interfacing and Applications, John Wiley and Sons, N.Y.

Birta, L.G., Abou-Rabia, O., 1983, "A Multi-microprocessor System for Parallel Solution of ODE's," SIAM 1983 Fall Meeting, Norfolk, Virginia.

Birta, L.G., Abou-Rabia, O., 1983, "A Multi-microprocessor System for Continuous System Simulation," Technical Report TR-83-08, Computer Science Department, University of Ottawa.

Birta, L.G., Abou-Rabia, O., 1984, "Performance of a Class of Parallel Methods for ODE's," Proceeding, 1984 European Simulation Meeting, Eger, Hungary, pp. 177-186.

Birta, L.G., Abou-Rabia, O., 1985, "Investigation of a Class of Parallel Methods for ODE's," submitted for publication in IEEE trans. computers.

Blech, R.A., Arpasi, D.J., 1981, "An Approach to Real-Time Simulation Using Parallel Processing," Proceeding, 1981 Summer Computer Simulation Conference, Washington, D.C., pp. 355-360.

Brundiers, H.J., et al., 1982, "The ETH-Multiprocessor EMPRESS: A Dynamically Configurable MIMD System," IEEE Transaction on Computers, Vol. C-31, No. 11, pp. 1035-1044.

Bursky, D., 1979, "Microprocessor Data Manual," Electronic Design, Vol. 24, Nov. 22, pp. 49-60.

Carnahan, B., Luther, H.A., Wilkes, J.O., 1969, Applied Numerical Methods, John Wiley & Sons, New York.

Cheng, S.C., Heng, K.T., Ng, W., 1977, "A Technique to Construct a Boiling Curve from Quenching Data Considering Heat Loss", Int. J. Multiphase Flow, Vol. 3, pp. 495-499.

Coffron, J.W., Long, W.E., 1983, Practical Interfacing Techniques for Microprocessor Systems, Prentice-Hall, Inc. Englewood Cliffs, N.J.

Collatz, L., 1966, "The Numerical Treatment of Differential Equations," Springer-Verlag, New York.

Cyre, W.R., et al., 1977, "WISPAC: A Parallel Array Computer for Large Scale System Simulation," Simulation, Vol. 29, No. 5, pp. 165-172.

Davis, H.A., 1979, "Comparing Architectures of Three 16-bit Microprocessors," Computer Design, Vol. 18, No. 7, pp. 91-100.

Davis, S., 1979, "Microprocessors," EDN, August 5, pp. 71-85.

Dekker, L., 1982, "A Missing Link to the Future of Simulation," SIMS 82 Annual Meeting, University of Trondheim, Norway.

Dekker, L., 1983, "Concepts for an Advanced Parallel Simulation Architecture," In: Simulation and Model-based Methodologies: An Integrated View, eds Oren, T.I., Ziegler, B.P., Elzas, M.S., Springer-Verlag, Heidelberg, pp. 235-278.

Dennis, J., 1980, "Data Flow Supercomputers," Computer, Vol. 13, No. 11, pp. 48-56.

Despain, A., Patterson, D., 1978, "X-Tree a Structured Multiprocessor Computer Architecture," Proceedings of the 5th Annual Symposium on Computer Architecture, Silver Spring, Md., IEEE Computer Society Press, pp. 144-151.

Enslow, P.H. JR., 1977, "Multiprocessor Organization - A Survey," Computing Survey, Vol. 9, No. 1, March, pp. 104-129.

Ercegovac, M.D., Karplus, W.J., 1984, "A Dataflow Approach in High-Speed Simulation of Continuous Systems," Proceeding, International Workshop on High-Level Computer Architecture 84, Los Angeles.

Ercegovac, M.D. et al., 1984, "Task Partitioning, Allocation and Simulation for a Dataflow Multiprocessor System," Proceeding, 1984 Summer Computer Simulation Conference, Boston, MA, pp. 326-331.

Fogarty, L.E. and Howe, R.M., 1968, "Computer Mechanization of Six-Degree of Freedom Flight Equations", Simulation, Vol. 11, No. 4, pp. 187-193.

Franklin, M.A., 1978, "Parallel Solution of Ordinary Differential Equations," IEEE Transaction on Computers, Vol. C-27, No. 5, pp. 413-420.

Garrow, R., et al., 1978, "16-bit Single Board Computer Maintains 8-bit Family Ties," Electronics, Vol. 51, No. 21, pp. 105-110.

Gear, C.W., 1971, Numerical Initial Value Problems in ODE's, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Gilbert, E.O., Howe, R.M., 1978, "Design Considerations in a Multiprocessor Computer for Continuous System Simulation," Proceeding, 1978 National Computer Conference, Vol. 47, pp. 385-393, AFIPS Press.

Goke, R., Lipovski, G.J., 1973, "Banyan Networks for Partitioning on Multiprocessor Systems," Proceeding, The 1st Annual Symposium on Computer Architecture, Silver Spring, Md., IEEE Computer Society Press, pp. 21-30.

Guarini, M., Cipriano, A., 1984, "A Microcomputer Based Parallel Dynamic System Simulator," Proceeding, 1984 Summer Computer Simulation Conference, Boston, MA., pp. 201-206.

Halin, H.J., et al., 1980, "The ETH Multiprocessor Project: Parallel Simulation of Continuous Systems," Simulation, Vol. 35, No. 4, pp. 109-123.

Hall, G., Watt, J.M., 1976, Modern Numerical Methods for Ordinary Differential Equations, Clarendon Press, Oxford.

Hamming, R.W., 1959, "Stable Predictor-Corrector Methods for Ordinary Differential Equations," Journal of the ACM, Vol. 6.

Hamming, R.W., 1973, Numerical Methods for Scientists and Engineers, McGraw-Hill, (second edition, pg. 400).

Havranek, W.A., 1977, "A New Digital Peripheral Computer for Simulation Problems and its Influence on Real-Time Computer Techniques," Parallel Computers-Parallel Mathematics, North-Holland Publishing Company, Amsterdam, pp. 197-202.

Haynes, L.S., Lau, R.L., Siewiorek, D.P., Mizell, D.W., 1982, "A Survey of Highly Parallel Computing," Computer, Vol. 15, No. 1, pp. 9-24.

Hemenway, J., Teja, E., 1979, "As you Get to Know the 8086, Use Your 8-bit Experience," EDN, January 20, pp. 81-87.

Henrici, P., 1962, Discrete Variable Methods in Ordinary Differential Equations, John Wiley & Sons, N.Y.

Heywood, S.A., 1983a, "The 8086-An Architecture for the Future, Part 1: Introduction and Glossary," Byte, June, pp. 450-455.

Heywood, S.A., 1983b, "The 8086-An Architecture for the Future, Part 2: Instruction Set," Byte, July, pp. 299-320.

Heywood, S.A., 1983c, "The 8086-An Architecture for the Future, Part 3: Instruction Set Continued," Byte, August, pp. 405-427.

Hull, T.E. et al., 1972, "Comparing Numerical Methods for Ordinary Differential Equatoins," SIAM Journal on Numerical Analysis, Vol. 9, No. 4, pp. 603-637.

Intel Corp., 1979a, Intel MULTIBUS Interfacing, Application Note AP-28A, Santa Clara, CA.

Intel Corp., 1981a, iSBC 86/05 Single Board Computer Hardware Reference Manual, Santa Clara, CA.

Intel Corp., 1981b, Intel MULTIBUS Specification, Santa Clara, CA.

Intel Corp., 1983a, Development Systems Handbook, Santa Clara, CA.

Intel Corp., 1983b, Memory Components Handbook, Santa Clara, CA.

Karplus, W.J., 1977, "Peripheral Processors for High-Speed Simulation," Vol. 29, No. 5, pp. 143-153.

Karplus, W.J., Makoui, A., 1982, "The Role of Data Flow Methods in Continuous Systems Simulation," Proceeding, 1982 Summer Computer Simulation Conference, Denver, Masachussetts, pp. 13-16.

Katz, B.J., et al., 1978, "8086 Microcomputer Bridges the Gap Between 8- and 16-bit Designs," Electronics, Vol. 51, No. 4, pp. 99-104.

Korn, G.A., 1972, "Back to Parallel Computation: Proposal for a completely New On-Line Simulation System Using Standard Mini-computers for Low-Cost Multiprocessing," Simulation, Vol. 19, No. 2, pp. 37-44.

Korn, C.A., Wait, J.V., 1978, Digital Continuous System Simulation, Prentice-Hall, Englewood Cliffs, New Jersey.

Krogh, F.T., 1973, "On Testing a Subroutine for the Numerical Integration of Ordinary Differential Equations," Journal of the ACM, Vol. 20, No. 4, pp. 545-562.

Krosel, S.M., Milner, E.J., 1982, "Application of Integration Algorithms in a Parallel Processing Environment for the Simulation of Jet Engines," Proceedings, Fifteenth Annual Simulation Symposium, Tampa, Florida.

Kung, H.T., 1976, "Synchronized and Asynchronous Parallel Algorithms for Multiprocessors," in: Algorithms and Complexity: New Directions and Recent Results, J.F. Traub, ed., Academic Press, N.Y., pp. 153-200.

Kung, H.T., 1982, "Why Systolic Architectures ?," Computer, Vol. 15, No. 1, pp. 37-46.

Lambert, J.D., 1973, Computational Methods in Ordinary Differential Equations, John Wiley, London.

Lapidus, L., Seinfeld, J.H., 1971, Numerical Solution of Ordinary Differential Equations, Mathematics in Science and Engineering, Vol. 74, Academic Press.

Lillevik, S.L., Easterday, J.L., 1983, " A Multiprocessor with Replicated Shared Memory," Proceedings of the National Computer Conference, AFIPS Press, pp. 557-564.

Locanthi, B.N., 1980, "The Homogeneous Machine," Technical Report 3759, Computer Science Department, California Institute of Technology.

McCullough, H.T., Lingaard, R.L., 1983, "A Special Purpose High Speed Computer for Digital Simulation," Proceeding, 1983 Summer Computer Simulation Conference, Vancouver, B.C., pp. 162-165.

McKevitt, J., Bayliss, J., 1979, "New Options from Big Chips," IEEE Spectrum, Vol. 16, No. 3, pp. 28-34.

McLean, A., 1980, "What's the Point in Simulation ?," SIMS meeting at Otaas, Finland.

Milne, W.E., 1953, Numerical Solution of Differential Equations, Wiley, N.Y.

Miranker, W.L., Liniger, W.M., 1967, "Parallel Methods for the Numerical Integration of Ordinary Differential Equations," Math. Comput., Vol. 21, pp. 303-320.

Miranker, W.L., 1971, "A Survey of Parallelism in Numerical Analysis," SIAM Review, Vol. 13, No. 4, pp. 524-547.

Morse, S.P. et al., 1978, "The Intel 8086 Microprocessor: A 16-bit Evolution of the 8080," Computer, Vol. 11, No. 6, pp. 18-27.

Ogdin, C.A., 1979, "Sixteen-Bit Micros," Mini-Micro Systems, January, pp. 64-72.

O'Grady, E.P., 1980, "A Communication Mechanism for Multiprocessor Simulation Systems," Simulation, Vol. 34, No. 2, pp. 39-49.

O'Grady, E.P., Wang, C.H., 1983, "Multibus-Based Parallel Processor for Simulation," Proceeding, 1983 Summer Computer Simulation Conference, Vancouver, B.C., pp. 371-375.

O'Grady, E.P., Wang, C.H., 1984, "Parallel-Processor Performance in a Jet-Engine Simulation," Proceeding, 1984 Summer Computer Simulation Conference, Boston, MA., pp. 311-316.

Rideout, V.C., et al., 1980, "Wisconsin Parallel Array Computer (WISPAC) Research Project," Reports UWECE 80-1, UWECE 80-27, UWECE 81-8, University of Wisconsin-Madison.

Rosser, J.B., 1967, "A Runge-Kutta for All Seasons," SIAM Review, Vol. 9, No. 3, pp. 417-452.

Rzehak, H., 1977, "Parallel Computers for Continuous Systems Simulation," Parallel Computers - Parallel Mathematics, North-Holland Publishing Company, Amsterdam, pp. 59-70.

Shampine, L.F., Watts, H.A., 1969, "Block Implicit One-Step Methods," Math. Comput., Vol. 23, pp. 731-740.

Shampine, L.F., Gordon, M.K., 1975, Computer Solution of Ordinary Differential Equations - The Initial Value Problem, W.H. Freeman and Co., San Francisco.

Shampine, L.F., Watts, H.A., Davenport, S.M., 1976, "Solving Non-stiff Ordinary Differential Equations - The State of the Art," SIAM Review, Vol. 18, No.3 , pp. 376-411.

Shampine, L.F., Watts, H.A., 1976, "Practical Solution of Ordinary Differential Equations by Runge-Kutta Methods," Sandia Laboratories Report SAND 76-0585.

Shannon, R.E., 1975, Systems Simulation: The Art and Science, Englewood Cliffs, N.J., Prentice-Hall Inc.

Siegel, H.J., 1979, "A Model of SIMD Machines and a Comparison of Various Interconnection Networks," IEEE Trans. Computers, Vol. C-28, No. 12, pp. 907-917.

Sips, H.J., 1982, "Philosophy Behind the Design and Construction of the Delft Parallel Processor as a Simulation Tool," SIMS 82 Annual Meeting," University of Trondhem, Norway.

Stone, H., 1971, "Parallel Processing with the Perfect Shuffle," IEEE trans. Computers, Vol. C-20, No. 2, pp. 153-161.

Stone, H.S., 1973, "Problems of Parallel Computation," in: Complexity of Sequential and Parallel Numerical Results, J.F. Traub, ed., Academic Press, N.Y., pp. 1-15.

Texas Instrument Incorporated, 1971, Designing with TTL Integrated Circuits, Texas Instrument Electronics Series, McGraw-Hill.

Texas Instrument Incorporated, 1976, The TTL Data Book for Design Engineers.

Verner, J.H., 1978, "Explicit Runge-Kutta Methods with Estimates of the Local Truncation Error," SIAM Journal on Numerical Analysis, Vol. 15, No. 4, pp. 772-790.

Watts, H.A., Shampine, L.F., 1972, "A-Stable Block Implicit One-Step Methods," BIT, Vol. 12, pp. 252-266.

Worland, P.B., 1976, "Parallel Methods for the Numerical Solution of Ordinary Differential Equations," IEEE Transactions on Computers, Vol. C-25, No. 10, pp. 1045-1048.

Yoshikawa, R., et al., 1977, "A Multi-Microprocessor Approach to a High Speed and Low-Cost Continuous System Simulation," Proceeding, National Computer Conference, AFIPS Press, pp. 931-936.

Zaks, R., Lesea, A., 1979, Microprocessor Interfacing Techniques, Third edition, Sybex Inc.

Zonneveld, J.A., 1964, "Automatic Numerical Integration," Mathematical Center Tracks, 8, Pub. Mathematisch Centrum Amsterdam.