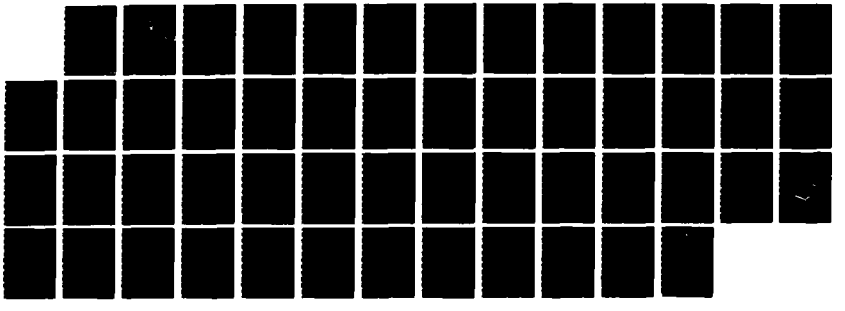
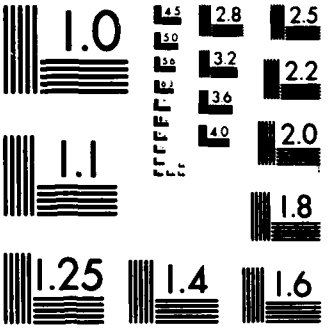
AD-A184 668 IMPLEMENTATION OF THE RUNGE-KUTTA-FEHLBERG METHOD FOR SOLUTION OF ORDINAR (U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA C F MAYO JUN 87 1/1

UNCLASSIFIED F/G 12/3 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

IMPLEMENTATION OF THE RUNGE-KUTTA-FEHLBERG METHOD FOR SOLUTION OF ORDINARY DIFFERENTIAL EQUATIONS ON A PARALLEL PROCESSOR

by

Colin F. Mayo

June 1987

Thesis Advisor: Charles E. Roberts, Jr.

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b OFFICE SYMBOL (If applicable) 53	7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	7b ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000
8a NAME OF FUNDING/SPONSORING ORGANIZATION 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)
IMPLEMENTATION OF THE RUNGE-KUTTA-FEHLBERG METHOD FOR SOLUTION OF ORDINARY DIFFERENTIAL EQUATIONS ON A PARALLEL PROCESSOR

12 PERSONAL AUTHOR(S)
Mayo, Colin F.

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year Month Day)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	1987 June	51

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Runge-Kutta-Fehlberg Method, Numerical Integration, Systems of Ordinary Differential Equations, Parallel Processor

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

A recent advance in computer architecture, the parallel processor computer, has made it theoretically feasible to reduce the time required to integrate a system of n ordinary differential equations by a factor of n. One established numerical technique, the Runge-Kutta-Fehlberg method, is adapted for parallel processing on an Intel Scientific Computer iPSC Concurrent Supercomputer. The algorithm is evaluated using a standardized collection of systems of equations. It is concluded that this type of parallel processor is not suited for the solution of this problem due to the communications overhead required. Short developments of ordinary differential equations and numerical integration methods are provided as background.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL Charles E. Roberts, Jr.	22b TELEPHONE (Include Area Code) (408) 646-2089 22c OFFICE SYMBOL 53Ro

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

Approved for public release; distribution is unlimited.

Implementation of the Runge-Kutta-Fehlberg Method
for Solution of Ordinary Differential Equations
on a Parallel Processor

by

Colin F. Mayo
Captain, United States Marine Corps
A.B., College of the Holy Cross, 1977

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN APPLIED MATHEMATICS

from the

NAVAL POSTGRADUATE SCHOOL
June 1987

Author: Colin F. Mayo
Colin F. Mayo

Approved by: Charles E. Roberts, Jr.
Charles E. Roberts, Jr., Thesis Advisor

Arthur L. Schoenstadt, Second Reader

Harold M. Fredricksen, Chairman,
Department of Mathematics

Kneale T. Marshall
Kneale T. Marshall
Dean of Information and Policy Sciences

ABSTRACT

A recent advance in computer architecture, the parallel processor computer, has made it theoretically feasible to reduce the time required to integrate a system of n ordinary differential equations by a factor of n. One established numerical technique, the Runge-Kutta-Fehlberg method, is adapted for parallel processing on an Intel Scientific Computer iPSC Concurrent Supercomputer. The algorithm is evaluated using a standardized collection of systems of equations. It is concluded that this type of parallel processor is not suited for the solution of this problem due to the communications overhead required. Short developments of ordinary differential equations and numerical integration methods are provided as background.

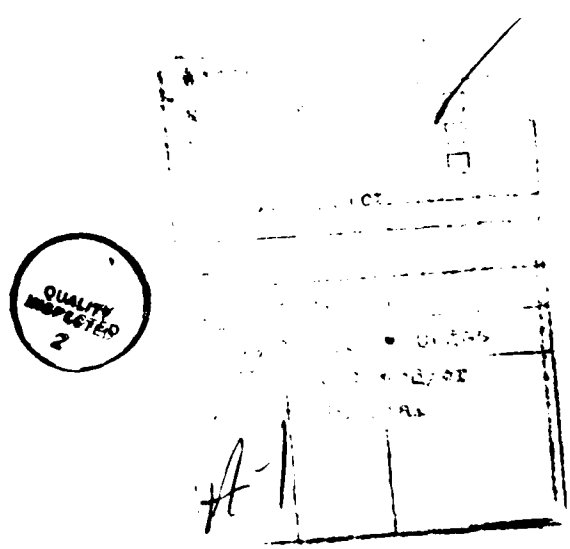


TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. PRELIMINARIES

A. INTRODUCTION

With the advent of modern computers, the implementation of numerical methods for the solution of systems of ordinary differential equations has become commonplace. Rather than developing new methods, the task at hand has become making current methods more efficient by reducing the amount of computer time needed to solve a system or by increasing the accuracy of the results. Previously, efforts to accomplish this task were centered on modifying existing algorithms.

A recent advance in computer architecture, the advent of parallel processing, has made it theoretically feasible to reduce the time required to integrate a system of n differential equations by a factor of n, assuming the parallel processor computer possesses n or more processors. This is a very significant time savings compared to those previously realized. This savings is provided by the parallel processor's ability to perform n different tasks, e.g. integration of n differential equations, simultaneously rather than sequentially as is done with a computer possessing a single processor.

In this thesis, one established method for solving systems of differential equations, the Runge-Kutta-Fehlberg method, is adapted for parallel processing on an Intel Scientific Computer iPSC Concurrent Supercomputer. The algorithm is then evaluated using a standardized collection of systems of equations. It is found, however, that this type of parallel processor is not suited for this purpose due to the communications overhead required. As background, short developments of ordinary differential equations and numerical methods are presented.

B. ORDINARY DIFFERENTIAL EQUATIONS

An equation that involves derivatives or differentials of a function or functions is called a differential equation. Differential equations are further classified as ordinary or partial differential equations. If the equation is a function of ordinary derivatives, it is an ordinary differential equation (ODE). If it is a function of partial derivatives, it is a partial differential equation (PDE). The order of a differential equation is the value of the highest derivative which appears in the differential equation. For the purposes of this thesis, the only concern is that of ordinary differential equations.

A first-order ordinary differential equation is the simplest case. Consider

$$y' = f(x,y). \tag{1.1}$$

This equation is a first-order ordinary differential equation. The general solution of 1.1 is a one-parameter family of curves. To select one member from this family, it is necessary to specify an initial value. That is to say the initial value of the dependent variable is specified for any value of the independent variable. An example of this would be

$$y' = f(x,y), \quad y(x_o) = y_o. \tag{1.2}$$

A system of n first-order equations is of the form

$$\begin{aligned} y_1' &= f_1(x, y_1, y_2, \ldots, y_n) \\ y_2' &= f_2(x, y_1, y_2, \ldots, y_n) \\ &\vdots \\ y_n' &= f_n(x, y_1, y_2, \ldots, y_n) \end{aligned} \tag{1.3}$$

where the y_i ($i = 1, 2, \ldots, n$) are functions of the independent variable x and f_i ($i = 1, 2, \ldots, n$) are functions of the $x, y_1, \ldots, y_n$. The solution to 1.3 will be a family of ordered n-tuples of the form

$$y = (y_1, y_2, \ldots, y_n). \tag{1.4}$$

Again, to select one member of this family of n-tuples, it is necessary to have an initial value. In this case, the initial value problem becomes a vector equation

$$y' = f(x,y), \quad y(x_o) = c_o. \tag{1.5}$$

An nth-order ordinary differential equation of the form

$$y^{(n)} = g(x, y, y^{(1)}, \ldots, y^{(n-1)}) \tag{1.6}$$

is solved by converting it into a set of n simultaneous first-order differential equations of the form

$$\begin{aligned} u_1' &= u_2 = f_1(x, u_1, \ldots, u_n) \\ u_2' &= u_3 = f_2(x, u_1, \ldots, u_n) \\ &\vdots \\ u_{n-1}' &= u_n = f_{n-1}(x, u_1, \ldots, u_n) \\ u_n' &= g(x, u_1, \ldots, u_n) \end{aligned} \tag{1.7}$$

by letting $u_1 = y$, $u_2 = y^{(1)}, \ldots, u_n = y^{(n-1)}$; by differentiating each of these equations: and by substituting for $y, y^{(1)}, \ldots, y^{(n-1)}$ in terms of $u_1, u_2, \ldots, u_n$. In order to determine a unique solution to this set of simultaneous equations, initial conditions must again be specified. These initial conditions are of the form $u_1(c) = d_1, u_2(c) = d_2, \ldots, u_n(c) = d_n$ which are obtained from transforming the conditions in terms of y and its derivatives. For further discussion of the solution of ordinary differential equations, see [Ref. 1].

Given a system of differential equations, the problem now becomes one of solving the system. Whenever possible, it is desirable to find an explicit solution. However, most systems cannot be solved exactly--that is, it is impossible to obtain a solution in elementary form. It is because of this characteristic of ordinary differential equations that we must turn to numerical methods to obtain the solutions.

II. DEVELOPMENT OF NUMERICAL METHODS

A. TAYLOR SERIES METHOD

Although the Taylor series methods are not usually used in practical problems, these methods must, however, be understood in order to understand the Runge-Kutta methods described in the next sections. In order to solve the initial value problem posed in Equation 1.2, we develop the relation between y and x by determining the coefficients of the Taylor series in which we expand y about the point $x = x_0$. If $y(x)$ has $m+1$ continuous derivatives on an interval I containing x_n, then by Taylor's Formula with remainder,

$$\begin{aligned} y(x) = y(x_n) + y'(x_n)(x-x_n) + (1/2!)y''(x_n)(x-x_n)^2 + \ldots \\ + (1/m!)y^{(m)}(x_n)(x-x_n)^m + (1/(m+1)!)y^{(m+1)}(c)(x-x_n)^{m+1}, \end{aligned} \tag{2.1}$$

for some $c \in (x,x_n)$. (Thomas and Finney [Ref. 2: pp. 663-665] show a detailed proof of this theorem.) If $y(x)$ is a solution to the initial value problem 1.2 and $y(x)$ has $m+1$ continuous derivatives, then

$$\begin{aligned} y'(x_n) &= f(x_n,y(x_n)) \\ y''(x_n) &= f^{(1)}(x_n,y(x_n)) = f_x + f_y y' = f_x + f_y f \\ y'''(x_n) &= f^{(2)}(x_n,y(x_n)) = f_{xx} + f_{xy}y' + (f_{yx} + f_{yy}y')y' + f_y y'' \\ &= f_{xx} + 2f_{xy}f + f_{yy}f^2 + f_x f_y + f_y^2 f, \end{aligned} \tag{2.2}$$

where f and its partial derivatives are all evaluated at $(x_n, y(x_n))$.

One could continue in this manner, computing any derivative of y evaluated at x_n, $y^{(m)}(x_n)$, in terms of f and its partial derivatives evaluated at x_n, $y(x_n)$. It is obvious to see that for other than reasonably small values of m, the derivatives are usually bothersome to compute. For this reason, m in Equation 2.1 is chosen to be reasonably small. By letting $h_n = (x-x_n)$, y_{n+1} is approximated by

$$\begin{aligned} y_{n+1} = y_n + f(x_n, y_n)h_n + (1/2!)f^{(1)}(x_n, y_n)h_n^2 + \ldots \\ + (1/m!)f^{(m-1)}(x_n, y_n)h_n^m. \end{aligned} \tag{2.3}$$

Equation 2.3 represents a single step numerical approximation to the solution of the initial value problem 1.2 and is known as the Taylor series method of order m.[1] From Equation 2.1, the error of this method is represented by

$$E_n = (1/(m+1)!)f^{(m)}(\xi, y(\xi))h_n^{m+1}, \tag{2.4}$$

where $\xi \in (x_n, x_{n+1})$.

As seen above, the computational disadvantages of the Taylor series method are due to the calculation and evaluation of the derivatives $f^{(1)}, f^{(2)}, \ldots, f^{(m-1)}$ at (x_n, y_n). It will be seen in the following sections that a Runge-Kutta method of order m is usually as accurate as the Taylor series of order m and is simpler to use. The Taylor series method, however, will be shown to be of theoretical value since the order of a Runge-Kutta method will be defined using the Taylor series method.

B. RUNGE-KUTTA METHOD

The German mathemetician Carl Runge (1856-1927) was the first to develop a numerical integration technique designed to approximate the Taylor series method without requiring explicit evaluations of derivatives beyond the first while preserving the accuracy of the Taylor series method [Ref. 3]. The technique was later improved by the German mathematician Martin Kutta (1867-1944) [Ref. 4] and, thus, it was named the Runge-Kutta method.

This technique sets up a problem with undetermined parameters and uses evaluations of f(x,y) within the interval (x_n, y_n) and (x_{n+1}, y_{n+1}) thus bypassing the derivatives of the Taylor series by requiring f(x, y) to be evaluated a number of additional times within the interval. The general form of this scheme is

$$y_{n+1} = y_n + \sum_{i=1}^{\nu} w_i k_i \tag{2.5}$$

where ν is the number of f(x, y) substitutions, w_i are the weighting coefficients, and the k_i satisfy the sequence

---

[1]Published by the English mathematician Brook Taylor (1685-1731) in 1715, however, Gregory and Leibnitz knew the series before Taylor, and John Bernoulli had published a similar result in 1694 [Ref. 1: p. 106].

$$
\begin{aligned}
k_1 &= hf(x_n, y_n) \\
k_2 &= hf(x_n + c_2 h, y_n + a_{21} k_1) \\
k_3 &= hf(x_n + c_3 h, y_n + a_{31} k_1 + a_{32} k_2) \\
&\vdots
\end{aligned}
\tag{2.6}
$$

The values of k can be thought of as estimates of the change in y when x changes a value of h. The problem now becomes one of determining the coefficients w_i, c_i, and a_{ij}. Each set of parameters will specify the points (x, y) where f(x, y) is to be evaluated. Therefore, this method calculates y_{n+1} using only a value for y_n and evaluations of f(x, y) at points between x_n and x_{n+1}. For this reason the method is termed *self-starting*. To obtain specific values for the coefficients, a value for **v** is chosen and y_{n+1} is expanded in powers of h such that it agrees as well as possible with the solution of the ordinary differential equation found using the Taylor series method. (For a complete development see [Ref. 5].)

A popular example of this method is the classical fourth-order Runge-Kutta method. It is given by

$$
\begin{aligned}
y_{n-1} &= y_n + (1/6)(k_o + 2k_1 + 2k_2 + k_3), \\
\text{where } k_o &= hf(x_n, y_n) \\
k_1 &= hf(x_n + h/2, y_n + k_o/2) \\
k_2 &= hf(x_n + h/2, y_n + k_1/2) \\
k_3 &= hf(x_n + h, y_n + k_2)
\end{aligned}
\tag{2.7}
$$

In this case, we avoid the derivatives of the Taylor series by performing four function evaluations on f(x, y) in the interval (x_n, y_n) and (x_{n+1}, y_{n+1}). As was stated earlier, the Taylor series method provides an error estimate for other methods. Here, as h goes to 0, this method agrees asymptotically with the Taylor series through the h^4 term, thus, making it a fourth-order method with error term proportional to E_5 from Equation 2.4. A disadvantage of this method is that an estimate of the local error is not readily available to help in choosing a suitable stepsize h.

C. RUNGE-KUTTA-FEHLBERG METHOD

In 1969, Erwin Fehlberg published a variation of the Runge-Kutta method which uses an estimate of local error to select a proper stepsize [Ref. 6]. For a given value of y_n, Fehlberg's method computes two estimates of y_{n+1} using fourth- and fifth-order Runge-Kutta formulas. In order to obtain an estimate of local error, the two values of y_{n+1} are compared. The stepsize is then adjusted, depending on the local error.

Fehlberg first uses

$$y_{n+1} = y_n + \sum_{i=1}^{6} c_i k_i \tag{2.8}$$

where the k_i satisfy

$$k_i = h_n f(x_n + \alpha_i h_n, y_n + \sum_{j=1}^{i-1} \beta_{ij} k_i) \quad i = 1, \ldots, 6. \tag{2.9}$$

This method requires six function evaluations per step. As with the fourth-order method discussed in Section II.B, the c_i are found by expanding y_{n+1} in powers of h_n so that it agrees as well as possible with the Taylor series solution. The coefficients determined by Fehlberg are found in Appendix A. Fehlberg found that the two expansions match until the h_n^6 term, thus, making the method fifth-order. This is a departure from the behavior of the nth-order Runge-Kutta methods where n = 1, 2, 3, 4 which produce (n+1)st order error. This partially explains the popularity of the classical fourth-order method described in the previous section; it takes two more function evaluations to obtain one more order of accuracy. Fehlberg's method, however, exploits the sixth function evaluation by determining a second value y_{n+1}^* using

$$y_{n+1}^* = y_n + \sum_{i=1}^{6} c_i^* k_i \tag{2.10}$$

This value was found to be fourth-order using the method described above. The local error is then estimated by

$$D_n = \sum_{i=1}^{6} (c_i - c_i^*)k_i \tag{2.11}$$

which is used for stepwise control. [Ref. 7: pp. 129-131]

Because the Runge-Kutta-Fehlberg method is generally thought of as one of the "best methods" available for solving nonstiff systems of equations [Ref. 8], it was chosen to adapt for parallel processing.

III. DESCRIPTION OF SEQUENTIAL PROGRAM

A FORTRAN program written by H.A. Watts and L.F. Shampine [Ref. 7: pp. 132-147] was chosen to be implemented. The program solves initial value problems in ordinary differential equations and is based on the Runge-Kutta-Fehlberg method described in Section II.C. It is designed to solve non-stiff and mildly stiff systems of differential equations when derivative evaluations are inexpensive. The program is typically used to integrate from a given initial value to a desired final value but can also be used as a one-step integrator. The program consists of a main program along with subroutines RKF45, RKFS, and FEHL. The following is a brief description of the program as it was developed for sequential processing.

A. MAIN PROGRAM

The main program first defines the system of equations to be solved through the use of the problem specific subroutine F. Additionally, it defines the system's initial conditions and program parameters such as absolute and relative error tolerances, and it provides output of data and error messages. Once the system of equations and parameters are defined, the main program begins solution of the problem by passing information to subroutine RKF45.

B. SUBROUTINE RKF45

Once the main program sets up the problem, subroutine RKF45 becomes the interfacing routine for the solution of the problem. RKF45 first sets up work arrays for storage of information used during integration, thus relieving the user of lengthy subroutine calling lists later in the program. It then calls subroutine RKFS, providing it with the work arrays.

C. SUBROUTINES RKFS AND FEHL

RKFS is the subroutine which, along with subroutine FEHL, performs the integration of the system of equations. It first establishes a minimum acceptable relative error and a maximum number of function evaluations allowed in order to avoid the expense of a user's attempt to obtain an excessive accuracy. It next checks input parameters, issuing error flags back to RKF45 as appropriate. Machine epsilon is then computed and used in conjunction with the minimum acceptable relative error to limit

precision difficulties. Error flags are issued if user specified relative error tolerance is too small. Once these preliminary tasks are complete, initialization is performed. This includes setting the function evaluation counter to zero and estimating the initial integration stepsize H. Throughout the program, the stepsize is not allowed to become smaller than 26 units of roundoff in the dependent variable T.

Once stepsize is computed and checked, subroutine FEHL is called. Subroutine FEHL contains the heart of the integrator in the form of the FORTRAN equivalent of the Runge-Kutta-Fehlberg formulas represented in Equations 2.8-2.10. Because of its importance, subroutine FEHL is included as Appendix B. FEHL performs the integration and returns to RKFS which in turn implements Equation 2.11 in order to determine local error and test to see if the integration step was successful. If unsuccessful, the stepsize is reduced and integration is attempted again. If successful, the solution at T+H is stored and the components of the system of equations are reevaluated at T+H using subroutine F. The function evaluation counter is changed to reflect the function evaluations performed in FEHL. This integration process continues until the final location is reached causing program flow to return to the main program which provides output of the final solution.

IV. IMPLEMENTATION

A. METHODOLOGY

In order to test the hypothesis that the time required to integrate a system of n ordinary differential equations can be reduced by a factor of n through parallel processing, the program described in Chapter III was implemented on an Intel Scientific Computer iPSC Concurrent Supercomputer. Appendix C contains a technical description of this computer. The particular computer used in this thesis possesses 16 processors thus making it a 16-node or 4-dimensional hypercube, as explained in the appendix.

The general scheme of the testing of this hypothesis was to choose systems to be integrated from a standard suite of problems used to test the performance of other integrator programs [Ref. 9: pp. 617-621]. Four systems were chosen in order to test performance of both small and moderate sized systems. As was done in the reference, the interval of integration for all implementations was [0, 20]. The constraint of only examining small and moderate sized systems was imparted due to the number of available independent processors being 16 or less. The systems chosen consist of 2 equation, 3 equation, 4 equation, and 10 equation systems. Each problem was first solved sequentially and timed on a single processor of the hypercube by adapting the code previously described, thus providing a sequential time standard to be compared with parallel run times. Next, the Runge-Kutta-Fehlberg algorithm was adapted for parallel processing using varying schemes to optimize performance of the hypercube. The systems were then solved using these schemes and timed. Detailed descriptions of each system's implementation are contained in this chapter following a discussion of internodal communication times.

B. INTERNODAL COMMUNICATION TIMES

The theoretical feasibility of reducing integration time by a factor of n assumes that the time required to pass information between processors is minimal when compared to the speed of computation. Time spent in communicating is a critical factor in the implementation of an integrator of systems of ordinary differential equations since, after each integration step, the solutions to each component of the system must be combined at a central location. This requires, for every step in the

integration, a message pass back to a central node from each node tasked with processing a separate component of the system of equations. For this reason, the hypercube's internodal communication times were empirically determined for later use in minimizing total time spent in communication when implementing the parallel algorithms.

Based on the topology of the 4-dimensional hypercube, as is discussed in Appendix C, it was decided to determine communication time for a message sent round trip from the host to the cube as well as between two nodes of distance 1, 2, 3, and 4 from each other. For these timings, a message of length 4 bytes was sent round trip 1000 times and an average round trip time was calculated. This experiment was performed a total of ten times for each and a final average round trip time was found. From the host to the cube, average round trip time was .02308 seconds. Average round trip times between nodes whose internodal distances are 1, 2, 3, and 4 were .002709 seconds, .005317 seconds, .006615 seconds, and .007797 seconds respectively. In addition, message passing was also timed for messages of length 16, 32, 40, and 80 bytes. These times were similar to those found using the 4 byte long message, thus showing that internodal communication times are not a function of message length.

Three conclusions may be drawn from these results. First, when minimization of communications time is desired, the host should be used only to house the main program and provide input and output. It should not be used as a "seventeenth" node due to the high relative order of host to cube communication time as compared with internodal communication times. Secondly, to minimize the total communication time in a given parallel algorithm, internodal distances must be considered when assigning tasks to specific nodes. Thus, in order to attain minimum program run times, parallel algorithms must create an optimum hypercube topology for a given system of equations based on internodal distances. Thirdly, since communication time is not message length dependent, it can also be concluded that a single long message is preferred to several short messages containing the same information.

C. A TWO EQUATION SYSTEM

1. System Description

Equation 4.1 depicts the two equation system chosen. It represents the growth of two conflicting populations.

$$\begin{aligned} y_1' &= 2(y_1 - y_1y_2), \qquad y_1(0) = 1, \\ y_2' &= -(y_2 - y_1y_2), \qquad y_2(0) = 3. \end{aligned} \tag{4.1}$$

2. **Sequential Implementation**

Sequential implementation was accomplished by adapting the integrator program described in Chapter III to run at node 0. This adaptation includes a subroutine F tailored to Equation 4.1. A main program, running at the host, loaded the integrator program to node 0 and provided output of integration results and sequential run times. It should be noted that these run times do not include time necessary to load the integrator program to node 0.

3. **Parallel Implementation I**

The parallelization scheme thought to be most natural, i.e. sending component 1 to node 1, component 2 to node 2, ... , component n to node n, was next implemented. This scheme was termed "shotgun" and consists of a main program at the host, the modified integrator program less subroutine FEHL at node 0, and node programs at nodes 1 and 2. Excerpts from the node 0 program and the node 1 and node 2 programs are listed in Appendix D.

Again, the main program loads the node programs and outputs integration results and parallel run times. The node 0 program is the driver program for the integration. It has been modified by removing subroutine FEHL and adding communications with nodes 1 and 2 which evaluate components 1 and 2 of the system respectively. It should be noted that these node assignments were made in order to insure internodal distances were minimized. Referring to Figure 4.1, the reason that this process was termed "shotgun" was due to the fact that the node 0 program sends and receives information from the nodes processing the component computations in a "shotgun" fashion.

A typical integration step takes place as in the sequential program except that instead of calling subroutine FEHL, the node 0 program first sends the component nodes the initial y vector. Although this message passing is sequential, the computation at the nodes does overlap, providing concurrent component processing. The component nodes perform the first function evaluation, using subroutine FNODE, and the first Runge-Kutta-Fehlberg step. This information is then sent back to node 0. This process takes place five times per integration step. Node 0 then computes a solution at the new location T + H, computes an appropriate stepsize, and again calls the component nodes to continue integration.

In general, the "shotgun" scheme may be extended to a 16-node hypercube to integrate systems up to size fifteen. For this scheme, the cost in number of message transmissions is represented by Equation 4.2 .

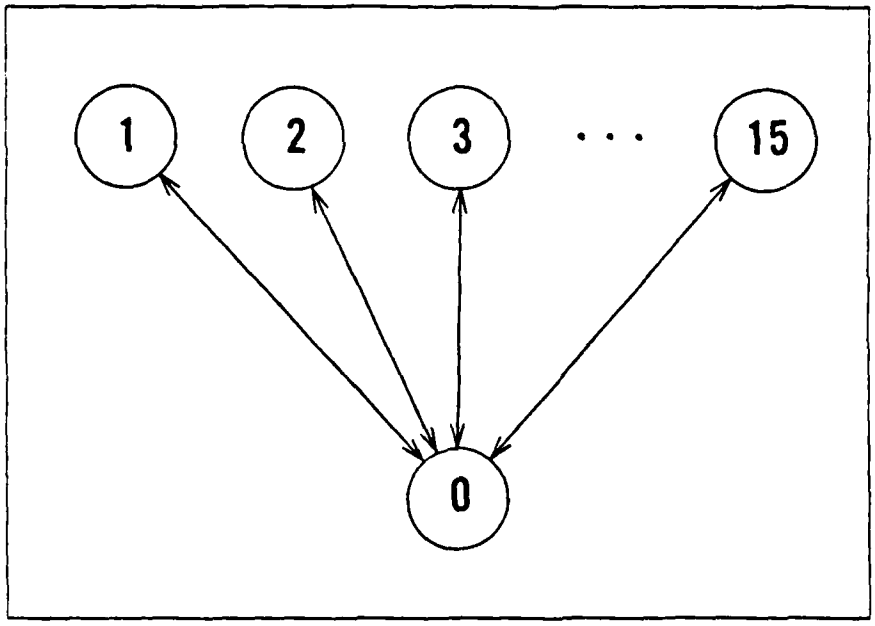


Figure 4.1 Shotgun Scheme.

$$COST = 2 \times NEQN \times NFE \tag{4.2}$$

where NEQN is the size of the system and NFE is the number of function evaluations. Since the number of function evaluations increases as the error tolerances are decreased, integration times can be expected to increase due both to the increased number of computations required, as well as the increased communications overhead requirement. For this system of two equations, the communications overhead in terms of the number of message passes performed is four times the number of function evaluations.

4. **Parallel Implementation II**

In order to reduce the communications overhead present in the "shotgun" implementation, a second integration scheme was developed. This new scheme, termed "flip-flop", involves sending information from node 0 to nodes 1 and 3 and then performing the integration step through a series of computations at these two nodes and message passes between them. Figure 4.2 depicts this scheme and program excerpts are contained in Appendix E.

Again the program at the host provides loading of the three node programs and output of the results. The node 0 program is the driver for the integration. The

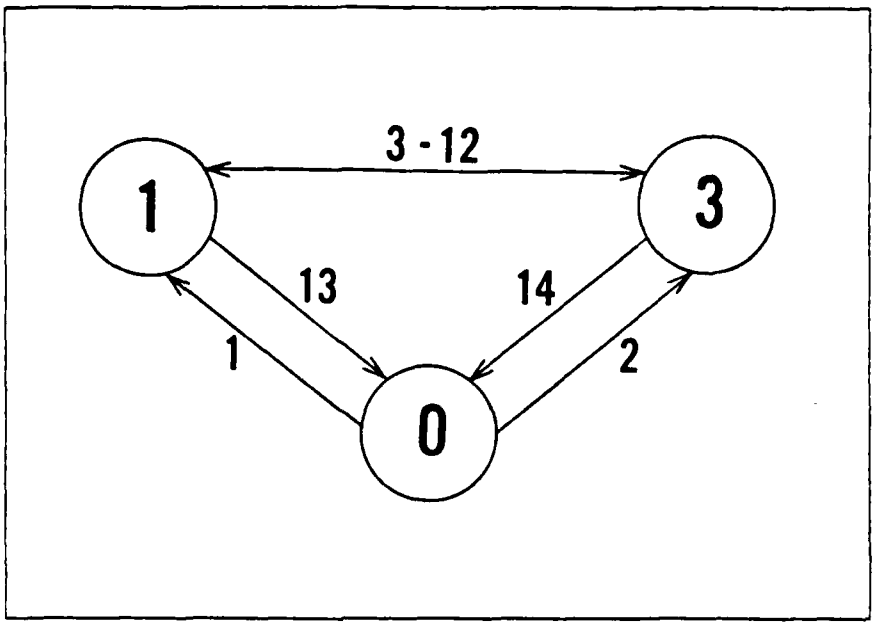


Figure 4.2 Flip-flop Scheme.

integrator segment of the original sequential code was again modified. It now only sends and receives one message from nodes 1 and 3 for a total of four message passes per integration step. Node 1 computes function evaluations for the first component, node 3 for the second. First, node 0 sends the initial information to each component node and these nodes compute the first function evaluation for the respective component. Once completed, the two component nodes exchange information in a "flip-flop" manner and then proceed with their respective second function evaluation. This process continues until all of the function evaluations in the Runge-Kutta-Fehlberg scheme are completed at the component nodes, at which time nodes 1 and 3 transmit their final component solutions back to node 0. The node 0 integrator program proceeds by advancing the integration as was done in previous programs.

In terms of communication overhead, the "flip-flop" scheme is superior to the "shotgun" scheme. The cost, in terms of message transmissions, is represented by Equation 4.3 .

$$COST = 2.8 \times NFE \tag{4.3}$$

where NFE is the number of function evaluations. The cost is derived from the fact that fourteen message transmissions occur during the computation of five function evaluations. These include ten between nodes 1 and 3, two between node 0 and node 1, and two between node 0 and node 3, as shown in Figure 4.2 . This cost is thirty percent of the cost of the "shotgun" algorithm for the two equation system.

D. A THREE EQUATION SYSTEM

1. System Description

The three equation system chosen is depicted in Equation 4.4 and represents a linear chemical reaction.

$$\begin{aligned} y_1' &= -y_1 + y_2, & y_1(0) &= 2, \\ y_2' &= y_1 + 2y_2 + y_3, & y_2(0) &= 0, \\ y_3' &= y_2 - y_3, & y_3(0) &= 1. \end{aligned} \tag{4.4}$$

2. Sequential Implementation

Sequential implementation was performed in the same manner described for the two equation system and by modifying the subroutine F to reflect Equation 4.4 .

3. Parallel Implementation I

The three equation system was first run parallel using the "shotgun" integration scheme with node 0 for the integrator program and nodes 1, 2, and 4 for the component programs. Nodes 1, 2, and 4 were chosen to again minimize internodal distances. Program excerpts are not included for this implementation as they are minor modifications of those found in Appendix D. From Equation 4.2, it can be determined that the the cost of solving the three equation system by the "shotgun" method is equal to six times the number of function evaluations.

4. Parallel Implementation II

An additional scheme was developed to decrease the total integration time for the three equation system. It was termed the "train" scheme due to its use of messages in the form of a real valued vector of size seventeen which is passed from node to node during integration. The vector contains information necessary for integration including values for T, stepsize H, y values, y' values and computed derivatives. Upon the message's arrival at a node, the node performs function evaluations for its respective component, updates information in the message, and passes it on. An analogy can be made between the process of updating information in the message and filling cars in a train; thus the name "train."

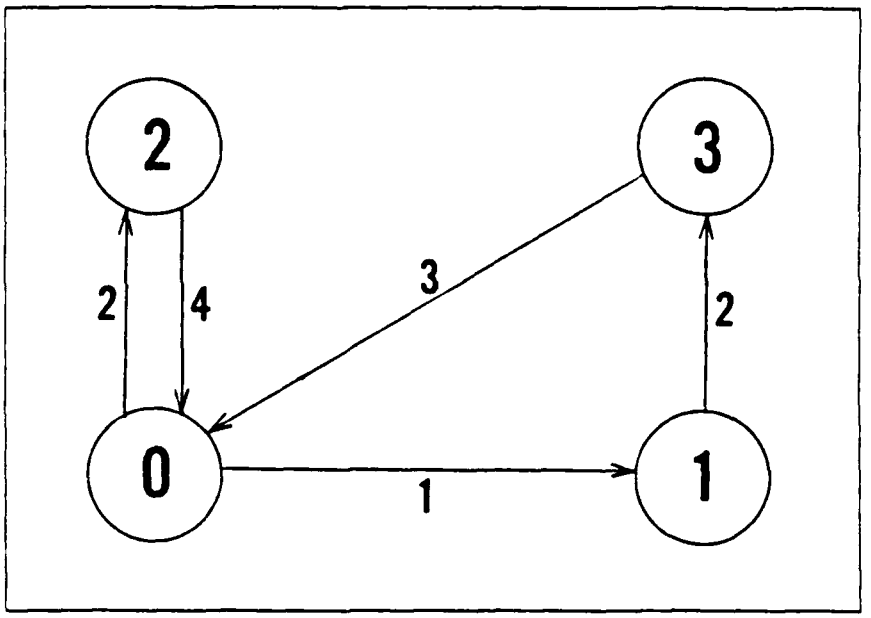


Figure 4.3 Three Equation Train Scheme.

Figure 4.3 depicts this integration scheme and program excerpts are listed in Appendix F. Node 0 is used to run the driver program while the three components of the system are processed at nodes 1, 2, and 3. Nodes 1 and 2 were chosen since they are of minimal distance to node 0, whereas, node 3 was chosen for its adjacency to node 1.

Referring to Figure 4.3 , the "train" scheme will be explained in terms of message pass time frames, meaning the time frame during which a message is passed between two nodes. In the first time frame, node 0 computes the first Runge-Kutta-Fehlberg function evaluation and sends the information "train" to node 1 which performs the second function evaluation. While this computation is being performed, the second time frame begins when node 0 sends the integration information to node 2. Upon completion of its computation, node 1 updates the information pertaining to its component and sends the "train" to node 3. At this point nodes 2 and 3 are performing their computations for the first function evaluation. As a worst case, it is assumed that node 3 sends its updated information to node 0 during the third time frame and that node 2 returns its information during a fourth time frame. Throughout this process, as information is received at node 0, the updated values are placed in a buffer until all component nodes return their updates. This completes one function

evaluation which takes a maximum of four message pass time frames during which five message passes occurred. This cycle is repeated until six function evaluations have been performed. Upon completion of the function evaluations, node 0 selects a new stepsize H and continues integration at the new location T + H.

In this topology, for one function evaluation, five message passes have been accomplished in the time associated with four. This gives a cost, in terms of message transmissions, as

$$COST = 4 \times NFE \tag{4.5}$$

where NFE is the number of function evaluations. This value is two thirds of the cost associated with the "shotgun" scheme for the three equation system.

E. A FOUR EQUATION SYSTEM

1. System Description

The four equation system implemented is a two body orbit problem and is represented in Equation 4.6 .

$$\begin{aligned} y_1' &= y_3, & y_1(0) &= 1 - \varepsilon, \\ y_2' &= y_4, & y_2(0) &= 0, \\ y_3' &= -y_1 (y_1^2 + y_2^2)^{3/2}, & y_3(0) &= 0, \\ y_4' &= -y_2 (y_1^2 + y_2^2)^{3/2}, & y_4(0) &= ((1 + \varepsilon)(1 - \varepsilon))^{1/2}, \end{aligned} \tag{4.6}$$

$\varepsilon = .9$, where ε is the eccentricity of the orbit.

2. Sequential Implementation

Sequential implementation was performed in the same manner as described for the two equation system and by modifying the subroutine F to reflect Equation 4.6 .

3. Parallel Implementation I

The four equation system was first run parallel using the "shotgun" integration scheme with node 0 for the integrator program and nodes 1, 2, 4, and 8 for the component programs. Again, these nodes were chosen to minimize internodal distances. Program excerpts are not included for this implementation as they are minor modifications of those found in Appendix D. From Equation 4.2, the cost of solving the four equation system with the "shotgun" method is equal to eight times the number of function evaluations.

4. **Parallel Implementation II**

The four equation system was also run using the "train" scheme. As shown in

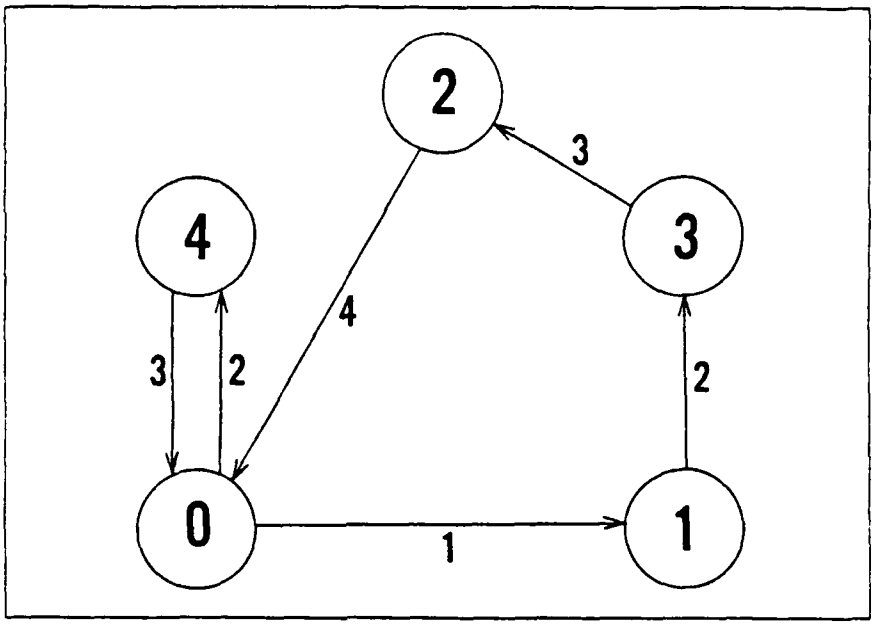


Figure 4.4 Four Equation Train Scheme.

Figure 4.4, this application employs nodes 0 through 4. Nodes 1, 2, and 4 were chosen for their adjacency to node 0 and node 3 was chosen for its adjacency to nodes 1 and 2. As in the three equation application, node 0 runs the driver program. Nodes 1 through 4 process their respective components as was done by the component nodes for the three equation system.

The integration process can again be described using message pass time frames. In the first time frame, node 0 makes the first Runge-Kutta-Fehlberg function evaluation and sends a vector message containing twenty-two pieces of information necessary for integration to node 1. Node 1 computes the second function evaluation for its component. In the second time frame, the "train" message is again sent out by node 1, this time to node 4. Node 4 in turn computes its function evaluation and sends updated information to node 0 where it is stored in a buffer until the other "train" message arrives. During the second time frame, node 1 also sends its updated message to node 3. In the third time frame, node 3 computes and sends its results to node 2. Finally, in the fourth time frame, node 2 computes new information and sends the

message containing updated information from nodes 1, 2, and 3 to node 0. In all, four message pass time frames have elapsed upon node 0 receiving all component information necessary for continuing integration. This cycle repeats until all six function evaluations are accomplished, causing node 0 to recompute stepsize H and continuing integration at T+H.

As with the three equation "train" implementation, the communications cost is

$$COST = 4 \times NFE \tag{4.7}$$

where NFE is the number of function evaluations. This is half the cost of the four equation "shotgun" scheme.

F. A TEN EQUATION SYSTEM

1. System Description

The radioactive decay chain problem listed in Equation 4.8 was chosen as the ten equation system.

$$\begin{aligned}
y_1' &= -y_1, & y_1(0) &= 1,\\
y_2' &= y_1 - y_2, & y_2(0) &= 0,\\
y_3' &= y_2 - y_3, & y_3(0) &= 0,\\
y_4' &= y_3 - y_4, & y_4(0) &= 0,\\
y_5' &= y_4 - y_5, & y_5(0) &= 0,\\
y_6' &= y_5 - y_6, & y_6(0) &= 0,\\
y_7' &= y_6 - y_7, & y_7(0) &= 0,\\
y_8' &= y_7 - y_8, & y_8(0) &= 0,\\
y_9' &= y_8 - y_9, & y_9(0) &= 0,\\
y_{10}' &= y_9, & y_{10}(0) &= 0.
\end{aligned} \tag{4.8}$$

2. Sequential Implementation

Sequential implementation was performed in the same manner as described for the two equation system and by modifying the subroutine F to reflect Equation 4.8 .

3. Parallel Implementation I

The ten equation system was first run parallel using the "shotgun" integration scheme with node 0 for the integrator program and nodes 1 through 10 for the component programs. In this case, internodal distances were not considered in the

node assignments. Program excerpts are not included for this implementation as they are minor modifications of those found in Appendix D. From Equation 4.2, the cost of solving the ten equation system with the "shotgun" method is equal to twenty times the number of function evaluations.

4. **Parallel Implementation II**

The ten equation system was also implemented using the message "train" technique. Figure 4.5 depicts the topology of the implementation. Nodes 0 through 10 were employed using node 0 as the driver and nodes 1 through 10 for component programs. Minimization of internodal distances was taken into account. As can be seen in Figure 4.5 , the four equation algorithm was modified to have three message "trains." Information from nodes 1, 2, and 3; nodes 4, 5, 6, and 7; and nodes 8, 9, and 10 is contained in each of the three messages. The computation flow is the same as the four equation algorithm except that node 1 transmits the message "train" to node 5 prior to performing its own function evaluation, then sends the updated message to node 3. Six time frames elapse during the course of a single function evaluation. During this period, thirteen message passes occur. The cost in communications overhead can be expressed as

$$COST = 6 \times NFE \tag{4.9}$$

where NFE is the number of function evaluations. When compared to the "shotgun" implementation, a seventy percent time savings is theoretically attained.

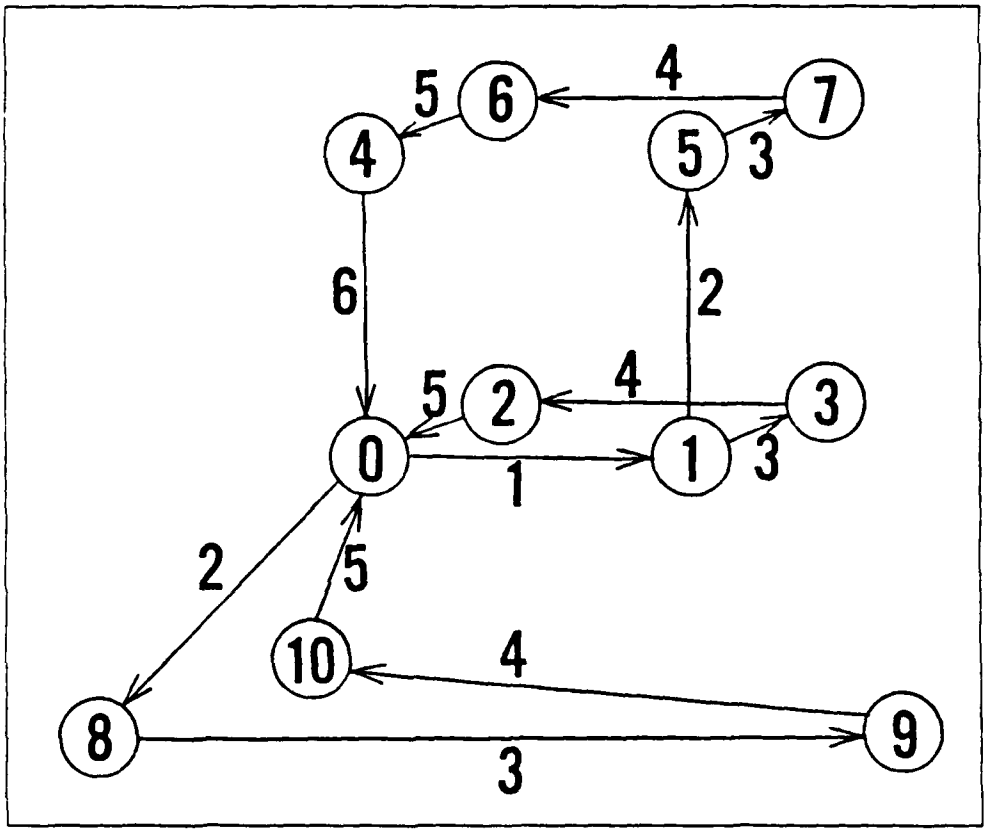


Figure 4.5 Ten Equation Train Scheme.

V. RESULTS AND CONCLUSIONS

A. RESULTS

Results of the sequential and parallel implementations described in Chapter IV are displayed graphically in Figures 5.1 through 5.4. For all implementations, the interval of integration was [0, 20], the absolute error tolerance set equal to 0.0, and the relative error tolerance was varied for each system as indicated below. In all four cases, the sequential implementation was fastest while the "shotgun" scheme was slowest. The "flip-flop" and "train" schemes resulted in reduced run times from those of the "shotgun" scheme. It is clear that these reductions, resulted from the lowering of the communications overhead required in the "shotgun" scheme.

1. Two Equation System

For all implementations, the two equation system was solved using relative error tolerances of 10^{-n} for n = 3, 4, ... , 8. The number of function evaluations performed was the same for all implementations and ranged from 403 to 2622 corresponding to relative error tolerances of 10^{-3} and 10^{-8} respectively. The "shotgun" scheme resulted in run times approximately eleven times longer than the sequential run times. The run times for the "flip-flop" implementation were approximately three and one half times those of the sequential runs and resulted in a seventy percent time savings over the "shotgun" times. This savings is in keeping with the comparison of equations 4.2 and 4.3 and it affirms that the communications overhead is the primary cause of parallel run times being slower than sequential run times for this algorithm.

2. Three Equation System

Relative error tolerances for the three equation system were successively set at 10^{-n} for n = 3, 4, ... , 7. The corresponding number of function evaluations ranged from 481 to 2418. Here, the "shotgun" run times were approximately twenty times the sequential run times, while the run times for the "train" scheme were approximately fourteen times the sequential run times. Again, the relationship between the communications overhead of the two parallel methods, as expressed in Equations 4.2 and 4.5, is upheld. The "train" scheme, in fact, attained run times that were about two thirds the run times for the "shotgun" scheme.

3. **Four Equation System**

The number of function evaluations performed in solving the four equation system ranged from 678 to 2771. These values corresponded to relative error tolerances ranging from 10^{-3} to 10^{-7}. "Shotgun" run times were approximately thirteen times sequential run times. Run times attained by the "train" scheme were seven to nine times those attained by the sequential runs and resulted in a seventy percent time savings over the "shotgun" scheme. This is a better result than encountered in the three equation "train" implementation because of the fact that the topologies of each scheme required the same number of message pass time frames for one function evaluation while the number of communications required in the "shotgun" scheme increased by two for the four equation system. Equations 4.2 and 4.7 predicted a fifty percent time savings going from the "shotgun" to the "train" implementation. The fact that the actual resulting savings was twenty percent better can be explained by looking at the derivation of Equation 4.7. It was derived using an upper bound of four message pass time frames when in fact the required number is likely to be somewhat less than the upper bound due to the semisequential nature of the "train" scheme's topology.

4. **Ten Equation System**

Finally, for the largest of the four systems implemented, relative error tolerances were varied from 10^{-3} through 10^{-10}. The minimum number of function evaluations was 252 corresponding to 10^{-3} while the maximum was 1639 for the error tolerance 10^{-10}. "Shotgun" timing results were approximately seventeen times the corresponding sequential results, whereas the "train" scheme results were a more respectable eight times the elapsed sequential run times. The savings over the "shotgun" results attained by the "train" scheme were from fifty-two to sixty percent of the "shotgun" times. These savings values are less than the theoretical savings predicted using Equations 4.2 and 4.9. Most likely, this disparity is due to the complexity of the topology of the ten equation "train" scheme as compared to the others. It is difficult to predict the time loss due to message collisions which occur at node 0 as the message "trains" return with their updated information. Therefore, the predicted number of elapsed time frames in Equation 4.9 is a "best guess" estimate and appears to be optimistic.

B. CONCLUSIONS

The theoretical reduction of times required to solve systems of ordinary differential equations by a factor of n was not attained through parallel processing. In fact, times required to solve systems of equations using parallel algorithms on the hypercube were greater than those of sequential algorithms. It was found that this is due to the communications overhead inherent in internodal message passing. When this high overhead is coupled with the requirement of numerical integration techniques to combine updated integration information from each system component after each function evaluation, parallel processing becomes ill suited. Two possible solutions to this problem might be (i) increased communications speeds within the hypercube or (ii) a small amount of available common memory for all the cube's nodes. This common memory would alleviate the need to artificially create it as was done by passing the "train" message from node to node and then transferring updated data in the "train" to a buffer at the driver node. The lack of any shared memory negated the concurrent processing of a system's components at the nodes by requiring message passes back to the driver node. Therefore, it is concluded that, due to the communications overhead encountered in conjuction with the manner in which systems of ordinary differential equations are numerically solved, parallel processors with totally distributed memory are not suited for the solution of systems of ordinary differential equations.

Additionally, several general conclusions may be drawn about parallel processing on the hypercube. First, the "natural" conversion of an existing sequential algorithm to a parallel algorithm may not be the best choice. This point was clearly supported in the failings of the "shotgun" implementations compared to the "train" implementations. Secondly, it is imperative to consider internodal adjacency and distances when developing parallel algorithms. This was evidenced by the empirical determination of internodal message pass times. Finally, parallel processing in itself is not a panacea. It is well suited and affords large time savings to many applications; however, it has been shown here that for at least one application, parallel processing causes a significant time loss over sequential processing.

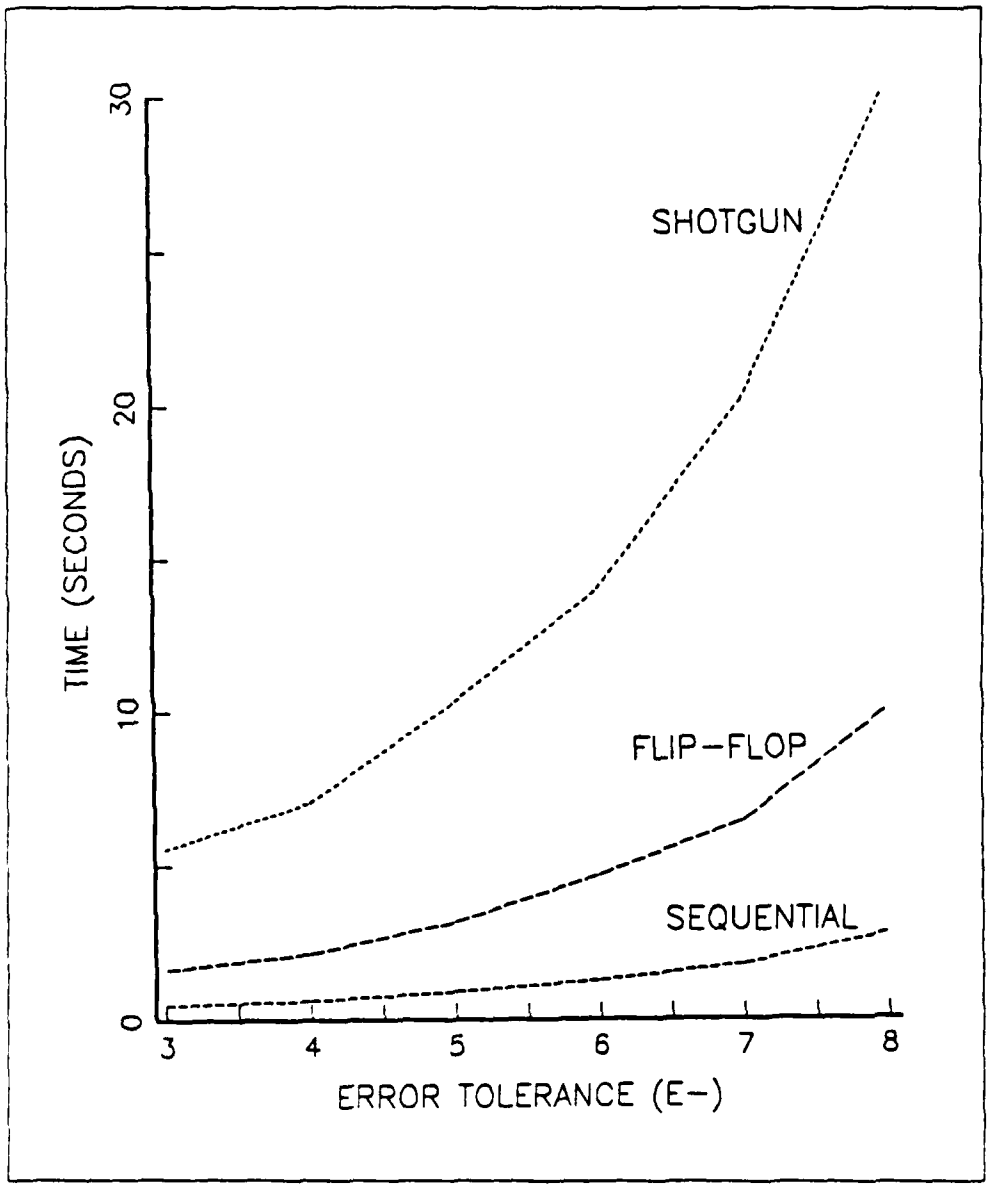


Figure 5.1 Results for the Two Equation System.

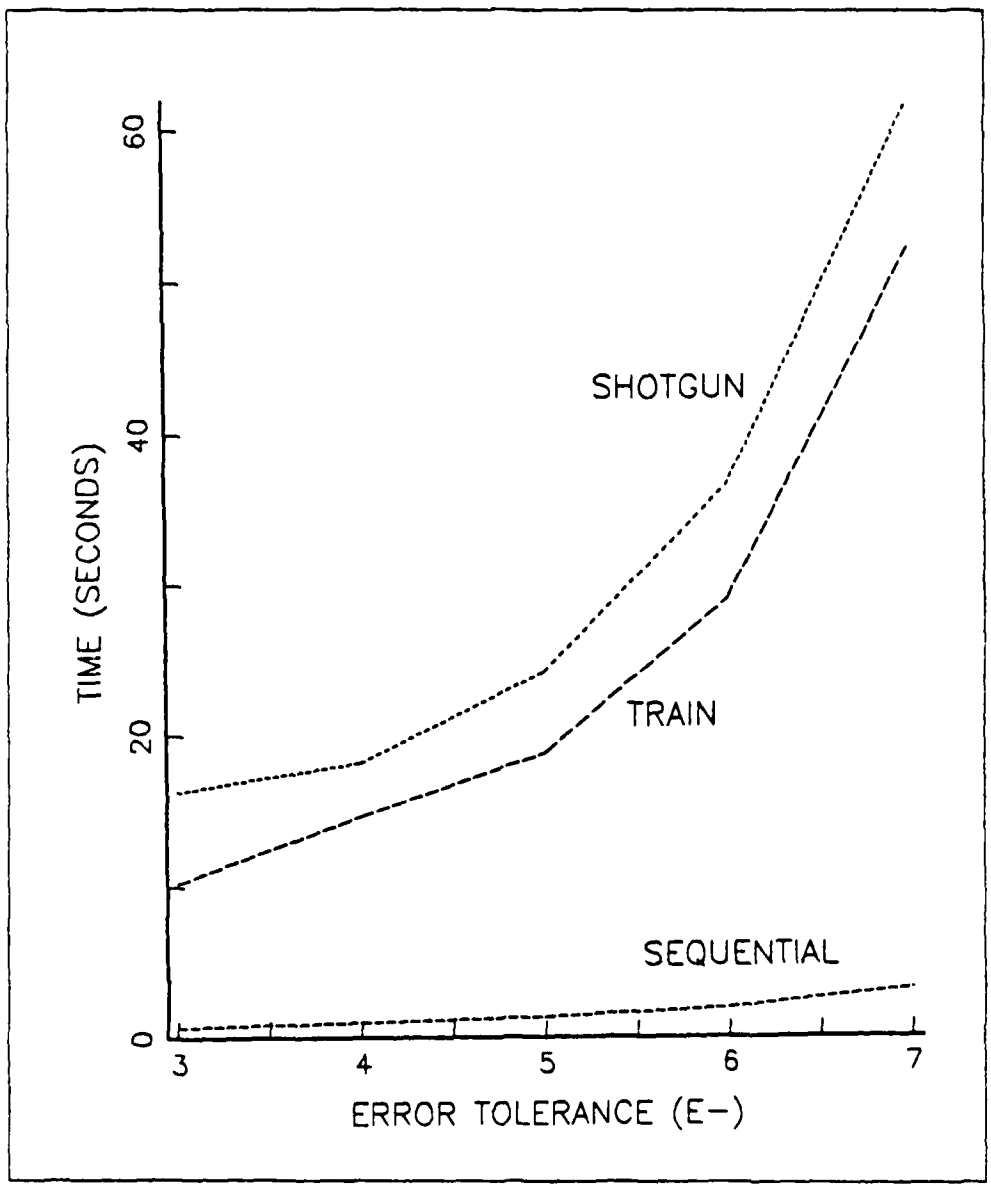


Figure 5.2 Results for the Three Equation System.

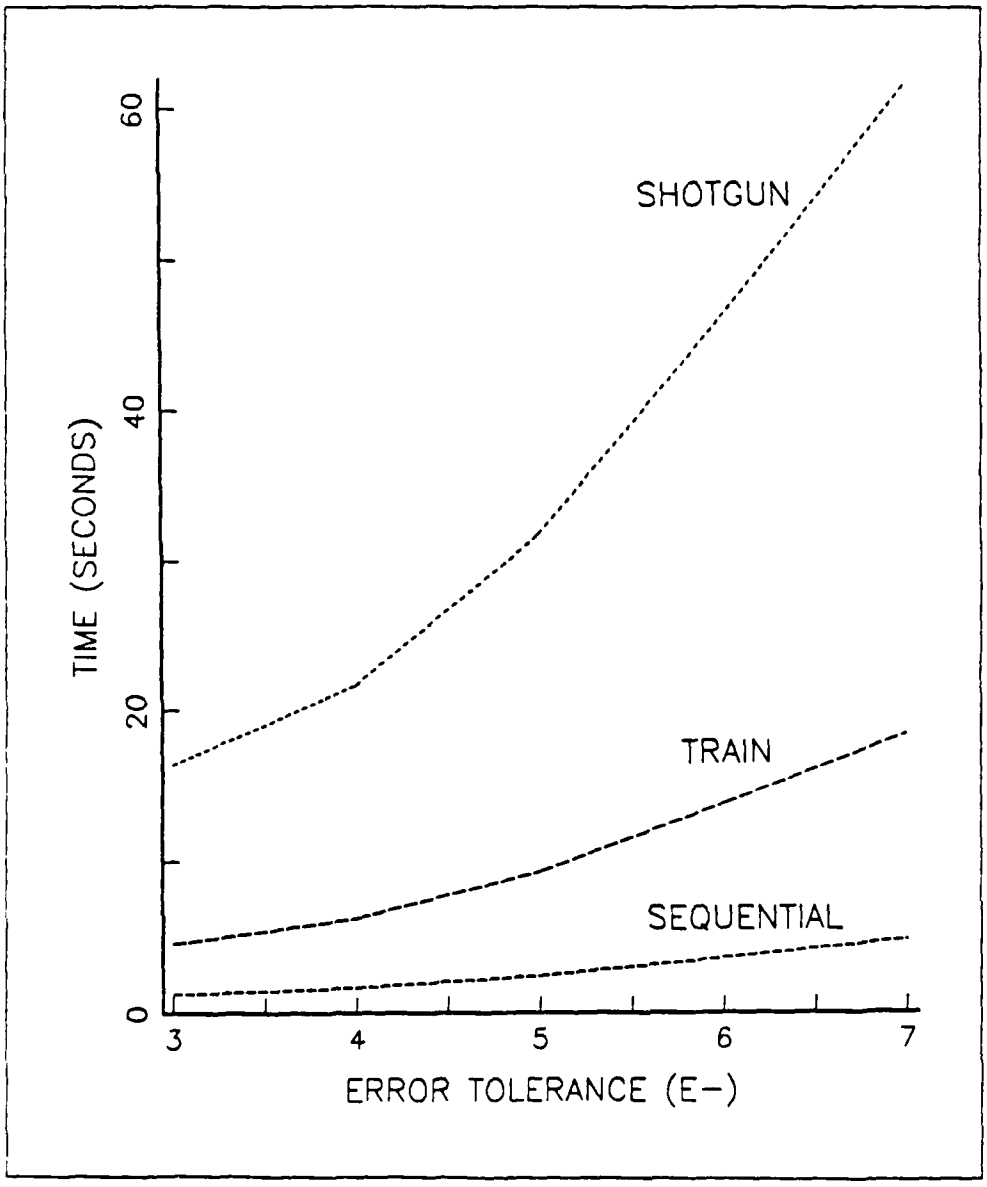


Figure 5.3 Results for the Four Equation System.

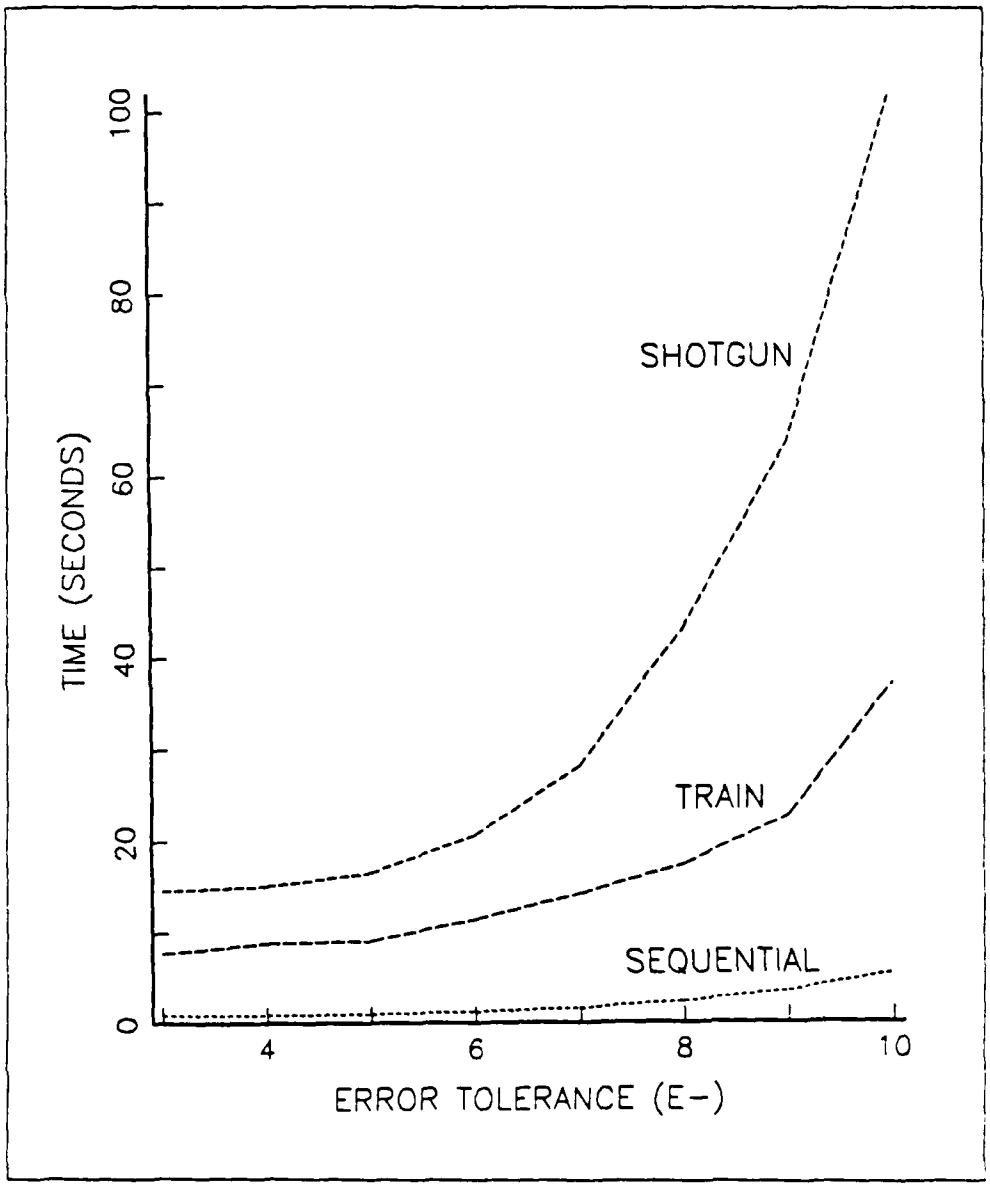


Figure 5.4 Results for the Ten Equation System.

APPENDIX A
RUNGE-KUTTA-FEHLBERG COEFFICIENTS

TABLE I
RUNGE-KUTTA-FEHLBERG COEFFICIENTS

α_i	β_{ij}					c_i	c_i^*
0						16/135	25/216
1/4	1/4					0	0
3/8	3/32	9/32				6656/12825	1408/2565
12/13	1932/2197	-7200/2197	7296/2197			28561/56420	2197/4104
1	439/216	-8	3680/513	-845/4104		-9/50	-1/5
1/2	-8/27	2	-3544/2565	1859/1404	-11/40	2/55	0

APPENDIX B
SUBROUTINE FEHL

```
      SUBROUTINE FEHL
C
C   FEHLBERG FOURTH-FIFTH ORDER RUNGE-KUTTA METHOD
C
C   FEHL INTEGRATES A SYSTEM OF NEQN FIRST ORDER
C   ORDINARY DIFFERENTIAL EQUATIONS OF THE FORM
C       DY(I)/DI=F(T,Y(1),...,Y(NEQN))
C   WHERE THE INITIAL VALUES Y(I) AND THE INITIAL DERIVATIVES
C   YP(I) ARE SPECIFIED AT THE STARTING POINT T.  FEHL ADVANCES
C   THE SOLUTION OVER THE FIXED STEP H AND RETURNS
C   THE FIFTH ORDER (SIXTH ORDER ACCURATE LOCALLY) SOLUTION
C   APPROXIMATION AT T+H IN ARRAY S(I).
C   F1,...,F5 ARE THE ARRAYS OF DIMENSION NEQN WHICH ARE NEEDED
C   FOR INTERNAL STORAGE.
C   THE FORMULAS HAVE BEEN GROUPED TO CONTROL LOSS OF SIGNIFICANCE.
C   FEHL SHOULD BE CALLED WITH AN H NOT SMALLER THAN 13 UNITS OF
C   ROUNDOFF IN T SO THAT THE VARIOUS INDEPENDENT ARGUMENTS CAN BE
C   DISTINGUISHED.
C
C
      INTEGER NEQN, K
      REAL Y(NEQN),T,H,CH,YP(NEQN),F1(NEQN),F2(NEQN),
     1  F3(NEQN),F4(NEQN),F5(NEQN),S(NEQN)
C
      CH=H/4.0
      DO 221 K=1,NEQN
  221 F5(K)=Y(K)+CH*YP(K)
      CALL F(T+CH,F5,F1)
C
      CH=3.0*H/32.0
      DO 222 K=1,NEQN
  222 F5(K)=Y(K)+CH*(YP(K)+3.0*F1(K))
      CALL F(T+3.0*H/8.0,F5,F2)
C
      CH=H/2197.0
      DO 223 K=1,NEQN
  223 F5(K)=Y(K)+CH*(1932.0*YP(K)+(7296.0*F2(K)-7200.0*F1(K)))
      CALL F(T+12.0*H/13.0,F5,F3)
C
      CH=H/4104.0
      DO 224 K=1,NEQN
  224 F5(K)=Y(K)+CH*((8341.0*YP(K)-845.0*F3(K))+
     1      (29440.0*F2(K)-32832.0*F1(K)))
      CALL F(T+H,F5,F4)
C
      CH=H/20520.0
      DO 225 K=1,NEQN
  225 F1(K)=Y(K)+CH*((-6080.0*YP(K)+((9295.0*F3(K)-
     1      5643.0*F4(K)))+(41040.0*F1(K)-28352.0*F2(K)))
      CALL F(T+H/2.0,F1,F5)
C
C   COMPUTE APPROXIMATE SOLUTION AT T+H
C
      CH=H/7618050.0
      DO 230 K=1,NEQN
  230 S(K)=Y(K)+CH*((902880.0*YP(K)+(3855735.0*F3(K)-
     1      1371249.0*F4(K)))+(3953664.0*F2(K)+
     2      277020.0*F5(K)))
C
      RETURN
      END
```

APPENDIX C
IPSC CONCURRENT SUPERCOMPUTER TECHNICAL DESCRIPTION

Implementation of the Runge-Kutta-Fehlberg method in this thesis was performed using an Intel Scientific Computer iPSC Concurrent Supercomputer. The basic system consists of two elements, a cube manager and a cube, as depicted in Figure C.1 .

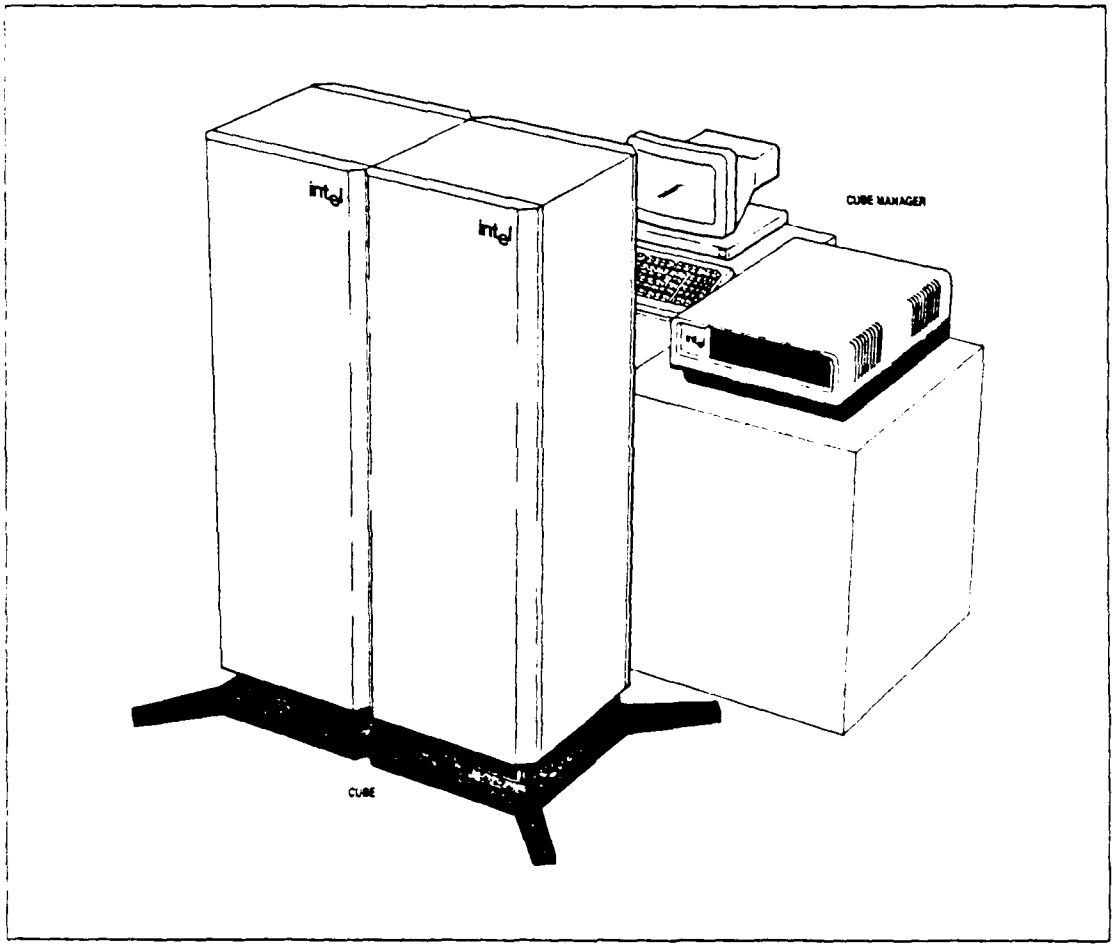


Figure C.1 32-Node iPSC Concurrent Supercomputer.

The cube manager is a desktop programming station that provides programming support and system management. It consists of an Intel System 310AP Multibus-

based computer using an Intel 80286 central processing unit and an Intel 80287 numeric processing unit. It also contains a 5 1/4" 140 megabyte Winchester disk, a 320K byte floppy disk, a 45 megabyte cartridge tape, and a 2 megabyte ECC RAM memory. Additionally, it is equipped with an integrated Ethernet interface for communicating with the cube, and an alphanumeric terminal for input/output. Cube manager software consists of a UNIX-based programming and development environment with FORTRAN, C, Assembler, cube control utilities and communications, and system diagnostics.

The cube is a complete ensemble of microcomputers connected in a parallel architecture. Each microcomputer, along with its own numeric processing unit and local memory is referred to as a "node." Nodes are connected together by high-speed communication channels to form a self-contained "cube" in a free-standing enclosure. Each node in the cube is an independent, single-board computer. The node contains an Intel 80286 central processing unit and its companion 80287 numeric processing unit. The node also contains 512K bytes of NMOS dynamic RAM and 8 bidirectional communication channels managed by dedicated 82586 communications coprocessors. Cube software consists of a monitor and kernal residing on each node. The monitor is contained in PROM and the kernal is loaded into node RAM after successful initialization. [Ref. 10]

The interconnection scheme, or topology, for the iPSC is a "binary n-cube" or "hypercube." The dimension n refers to the power of two corresponding to the number of nodes in the cube. In the case of the iPSC computer used in this thesis, the number of nodes is 16; thus, making it a 4-dimensional hypercube, as depicted in Figure C.2 .

Within a 4-dimensional hypercube, each node has 4 nodes adjacent to it. The distance between a node and one of its adjacent nodes is defined to be 1. Additionally, there are 6 nodes with distance 2, 4 nodes with distance 3, and 1 node with distance 4 from any given node in the 4-dimensional hypercube. These internodal distances must be considered when employing parallel algorithms, in order to minimize distances over which messages are passed.

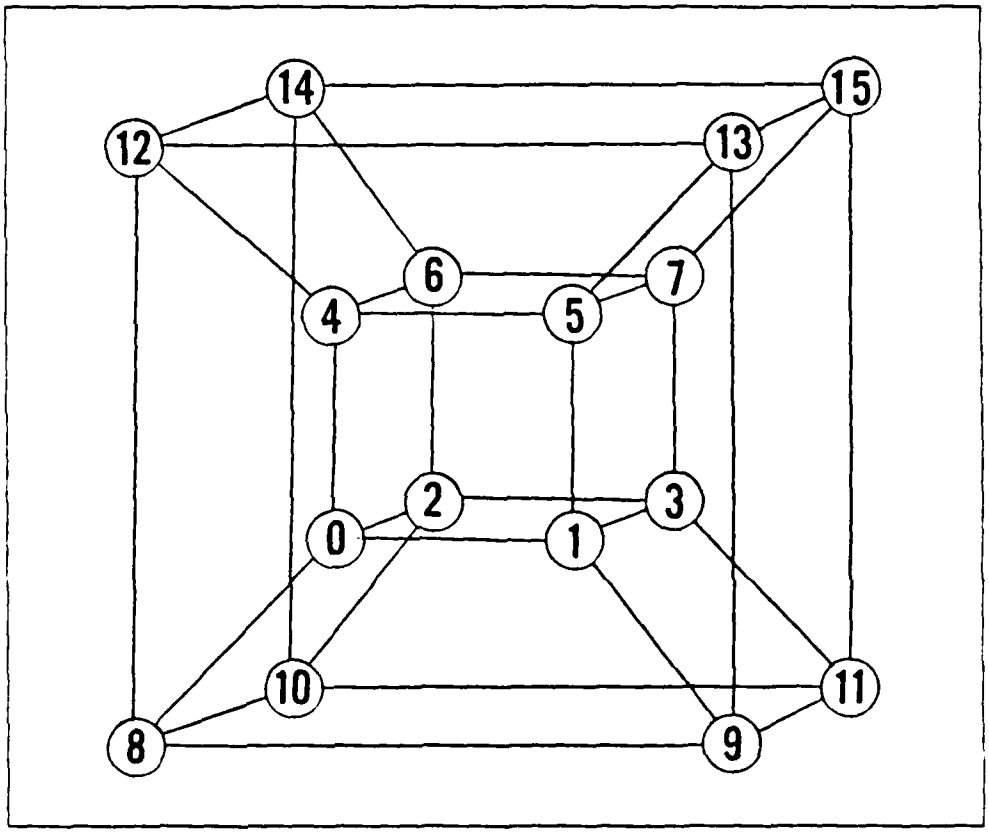


Figure C.2 4-Dimensional Hypercube Topology.

APPENDIX D

PROGRAM LISTING EXCERPTS FOR THE TWO EQUATION SHOTGUN SCHEME

The following listing is an excerpt from the node 0 program for the Shotgun implementation of the two equation system.

```
                          .
                          .
                          .
  220 buf(1)=t
      buf(2)=h
c
      ch=h/4.0
      do 221 k=1,2
       buf(3)=y(k)
       buf(4)=yp(k)
       call sendw(ci,10,buf,16,k,1)
  221 f5(k)=y(k)+ch*yp(k)
c
      do 222 k=1,2
  222  call sendw(ci,20,f5,len,k,1)
c
      do 223 k=1,2
       call recvw(ci,15,z,4,cnt,frnode,frpid)
  223  f5(frnode)=z
c
      do 224 k=1,2
  224  call sendw(ci,30,f5,len,k,1)
c
      do 225 k=1,2
       call recvw(ci,15,z,4,cnt,frnode,frpid)
  225  f5(frnode)=z
c
      do 226 k=1,2
  226  call sendw(ci,40,f5,len,k,1)
c
      do 227 k=1,2
       call recvw(ci,15,z,4,cnt,frnode,frpid)
  227  f5(frnode)=z
c
      do 228 k=1,2
  228  call sendw(ci,50,f5,len,k,1)
c
      do 229 k=1,2
       call recvw(ci,15,z,4,cnt,frnode,frpid)
  229  f1(frnode)=z
c
      do 230 k=1,2
  230  call sendw(ci,60,f1,len,k,1)
c
      do 231 k=1,2
       call recvw(ci,15,zz,8,cnt,frnode,frpid)
       f1(frnode)=zz(1)
  231  eee(frnode)=zz(2)
c
                          .
                          .
                          .
```

The following listing is the node 1 and node 2 program for the Shotgun implementation of the two equation system.

```
      program node2eq
c.....nodes 1 and 2.....2 equation system....
c
      integer chan,copen,nodeid,cnt,frnode
      dimension buf(4),d5(2),zz(2)
      equivalence (buf(1),t),(buf(2),h),(buf(3),a),buf(4),b)
c
      chan=copen(mypid())
      nodeid=mynode()
c
   10 call recvw(chan,10,buf,16,cnt,frnode,frpid)
      tp=t+h/4.0
      call recvw(chan,20,d5,8,cnt,frnode,frpid)
      call fnode(tp,d5,w1,nodeid)
      z=a+3.0*h*(b+3.0*w1)/32.0
c
      call sendw(chan,15,z,4,0,1)
      tp=t+3.0*h/8.0
      call recvw(chan,30,d5,8,cnt,frnode,frpid)
      call fnode(tp,d5,w2,nodeid)
      z=a+h*(1932.0*b+(7296.0*w2-7200.0*w1))/2197.0
c
      call sendw(chan,15,z,4,0,1)
      tp=t+12.0*h/13.0
      call recvw(chan,40,d5,8,cnt,frnode,frpid)
      call fnode(tp,d5,w3,nodeid)
      z=a+h*((8341.0*b-845.0*w3)+(2944.0*w2-32832.0*w1))/4104.0
c
      call sendw(chan,15,z,4,0,1)
      tp=t+h
      call recvw(chan,50,d5,8,cnt,frnode,frpid)
      call fnode(tp,d5,w4,nodeid)
      z=a+h*((-6080.0*b+(9295.0*w3-5643.0*w4))
     1    +(41040.0*w1-28352.0*w2))/20520.0
c
      call sendw(chan,15,z,4,0,1)
      tp=t+h/2.0
      call recvw(chan,60,d5,8,cnt,frnode,frpid)
      call fnode(tp,d5,w5,nodeid)
      zz(1)=a+h*((902880.0*b
     1     +(3855735.0*w3-1371249.0*w4))
     2     +(3953664.0*w2+277020.0*w5))/7618050.0
      zz(2)=abs((-2090.0*b+(21970.0*w3-15048.0*w4))
     1     +(22528.0*w2-27360.0*w5))
      call sendw(chan,15,zz,8,0,1)
c
      go to 10
c
      end
c
c
      subroutine fnode(tp,d,v,nodeid)
      dimension d(2)
c
      go to (1,2), nodeid
c
    1 v=2*(d(1)-d(1)*d(2))
      return
c
    2 v=-(d(2)-d(1)*d(2))
      return
c
      end
```

APPENDIX E

PROGRAM LISTING EXCERPTS FOR THE TWO EQUATION FLIP-FLOP SCHEME

The following listing is an excerpt from the node 0 program for the Flip-flop implementation of the two equation system.

```
                          .
                          .
                          .
  220 buf(1)=t
      buf(2)=h
c
      ch=h/4.0
c
      buf(3)=y(1)
      buf(4)=yp(1)
       call sendw(ci,10,buf,16,1,1)
      buf(3)=y(2)
      buf(4)=yp(2)
       call sendw(ci,10,buf,16,3,1)
c
      do 221 k=1,2
  221 f5(k)=y(k)+ch*yp(k)
c
      call sendw(ci,20,f5,len,1,1)
c
      do 231 k=1,2
         call recvw(ci,15,zz,8,cnt,frnode,frpid)
         if (frnode .eq.1) then
            f1(frnode)=zz(1)
            eee(frnode)=zz(2)
         else
            frnode=frnode-1
            f1(frnode)=zz(1)
            eee(frnode)=zz(2)
         end if
  231 continue
                          .
                          .
                          .
```

The following listing is an excerpt from the node 1 program for the Flip-flop implementation of the two equation system.

```
                          .
                          .
                          .
   10  call recvw(chan,20,d,8,cnt,frnode,frpid)
       w1=2.0*(d(1)-d(1)*d(2))
       z=a+3.0*h*(b+3.0*w1)/32.0
c
       call sendw(chan,23,buf1,12,3,1)
       call recvw(chan,32,buf1,12,cnt,frnode,frpid)
       tp=t+3.0*h/8.0
       w2=2.0*(d(1)-d(1)*d(2))
       d(1)=a+h*(1932.0*b+(7296.0*w2-7200.0*w1))/2197.0
       call recvw(chan,42,d(2),4,cnt,frnode,frpid)
       call sendw(chan,43,d(1),4,3,1)
       tp=t+12.0*h/13.0
       w3=2.0*(d(1)-d(1)*d(2))
```

```
      d(1)=a+h*((8341.0*b-845.0*w3)+(29440.0*w2-32832.0*w1))/4104.0
      call recvw(chan,42,d(2),4,cnt,frnode,frpid)
      call sendw(chan,43,d(1),4,3,1)
      tp=t+h
      w4=2.0*(d(1)-d(1)*d(2))
      d(1)=a+h*((-6080.0*b+(9295.0*w3-5643.0*w4))
     1     +(41040.0*w1-28352.0*w2))/20520.0
      call recvw(chan,42,d(2),4,cnt,frnode,frpid)
      call sendw(chan,43,d(1),4,3,1)
      tp=t+h/2.0
      w5=2.0*(d(1)-d(1)*d(2))
      zz(1)=a+h*((902880.0*b
     1      +(3855735.0*w3-1371249.0*w4))
     2      +(3953664.0*w2+277020.0*w5))/7618050.0
      zz(2)=abs((-2090.0*b+(21970.0*w3-15048.0*w4))
     1      +(22528.0*w2-27360.0*w5))
      call sendw(chan,15,zz,8,0,1)
c
      go to 10
                        .
                        .
                        .
```

The following listing is an excerpt from the node 3 program for the Flip-flop implementation of the 2 equation system.

```
                        .
                        .
                        .
   10 call recvw(chan,20,d,8,cnt,frnode,frpid)
      tp=t+h/4.0
      call recvw(chan,23,buf1,12,cnt,frnode,frpid)
      w1=-(d(2)-d(1)*d(2))
      d(2)=a+3.0*h*(b+3.0*w1)/32.0
      d(1)=buf1(3)
c
      call sendw(chan,32,buf1,12,1,1)
      tp=t+3.0*h/8.0
      w2=-(d(2)-d(1)*d(2))
      d(2)=a+h*(1932.0*b+(7296.0*w2-7200.0*w1))/2197.0
      call sendw(chan,42,d(2),4,1,1)
      call recvw(chan,43,d(1),4,cnt,frnode,frpid)
      tp=t+12.0*h/13.0
      w3=-(d(2)-d(1)*d(2))
      d(2)=a+h*((8341.0*b-845.0*w3)+(29440.0*w2-32832.0*w1))/4104.0
      call sendw(chan,42,d(2),4,1,1)
      call recvw(chan,42,d(1),4,cnt,frnode,frpid)
      tp=t+h
      w4=-(d(2)-d(1)*d(2))
      d(2)=a+h*((-6080.0*b+(9295.0*w3-5643.0*w4))
     1     +(41040.0*w1-28352.0*w2))/20520.0
      call sendw(chan,42,d(2),4,1,1)
      call recvw(chan,43,d(1),4,cnt,frnode,frpid)
      tp=t+h/2.0
      w5=-(d(2)-d(1)*d(2))
      zz(1)=a+h*((902880.0*b
     1      +(3855735.0*w3-1371249.0*w4))
     2      +(3953664.0*w2+277020.0*w5))/7618050.0
      zz(2)=abs((-2090.0*b+(21970.0*w3-15048.0*w4))
     1      +(22528.0*w2-27360.0*w5))
      call sendw(chan,15,zz,8,0,1)
c
      go to 10
                        .
                        .
                        .
```

APPENDIX F
PROGRAM LISTING EXCERPTS FOR THE THREE EQUATION TRAIN SCHEME

The following listing is an excerpt from the node 0 program for the Train implementation of the three equation system.

```
                        .
                        .
                        .
  220 buf(1)=t
      buf(2)=h
c
      ch=h/4.0
      do 222 k=1,3
       buf(k+2)=y(k)
       buf(k+5)=yp(k)
  222  buf(k+8)=f5(k)
c
       do 224 ijk=1,4
       call sendw(ci,10,buf,68,1,1)
       call sendw(ci,10,buf,68,2,1)
c
      do 223 ii=1,2
       call recvw(ci,10,buf1,68,cnt,frnode,frpid)
          if (frnode .eq. 2) then
            buf(10)=buf1(13)
          else
            buf(9)=buf1(12)
            buf(11)=buf1(14)
          end if
c
  223 continue
  224 continue
c
       call sendw(ci,10,buf,68,1,1)
       call sendw(ci,10,buf,68,2,1)
c
      do 225 ii=1,2
       call recvw(ci,10,buf1,68,cnt,frnode,frpid)
          if (frnode .eq. 2) then
            f1(2)=buf1(13)
            eee(2)=buf1(16)
          else
            f1(1)=buf1(13)
            f1(3)=buf1(14)
            eee(1)=buf1(15)
            eee(2)=buf1(17)
          end if
  225  continue
                        .
                        .
                        .
```

The following listing is the component node for the Train implementation of the three equation system.

```
                        .
                        .
                        .
       nodp2=nodeid+2
```

```
      nodp5=nodeid+5
      nodp11=nodeid+11
      nodp14=nodeid+14
c
   10 call recvw(chan,10,buf,len,cnt,frnode,frpid)
      tp=t+h/4.0
      a=buf(nodp2)
      b=buf(nodp5)
      call fnode(tp,d5,w1,nodeid)
      buf(nodp11)=a+3.0*h*(b+3.0*w1)/32.0
c
      call sendw(chan,10,buf,len,4,ndest,1)
      tp=t+3.0*h/8.0
      call recvw(chan,10,buf,len,cnt,frnode,frpid)
      call fnode(tp,d5,w2,nodeid)
      buf(nodp11)=a+h*(1932.0*b+(7296.0*w2-7200.0*w1))/2197.0
c
      call sendw(chan,10,buf,len,ndest,1)
      tp=t+12.0*h/13.0
      call recvw(chan,10,buf,len,cnt,frnode,frpid)
      call fnode(tp,d5,w3,nodeid)
      buf(nodp11)=a+h*((8341.0*b-845.0*w3)+(2944.0*w2-32832.0*w1))
     1     /4104.0
c
      call sendw(chan,10,buf,len,ndest,1)
      tp=t+h
      call recvw(chan,10,buf,len,cnt,frnode,frpid)
      call fnode(tp,d5,w4,nodeid)
      buf(nodp11)=a+h*((-6080.0*b+(9295.0*w3-5643.0*w4))
     1    +(41040.0*w1-28352.0*w2))/20520.0
c
      call sendw(chan,10,buf,len,ndest,1)
      tp=t+h/2.0
      call recvw(chan,10,buf,len,cnt,frnode,frpid)
      call fnode(tp,d5,w5,nodeid)
      buf(nodp11)=a+h*((902880.0*b
     1     +(3855735.0*w3-1371249.0*w4))
     2     +(3953664.0*w2+277020.0*w5))/7618050.0
      buf(nodp14)=abs((-2090.0*b+(21970.0*w3-15048.0*w4))
     1     +(22528.0*w2-27360.0*w5))
      call sendw(chan,10,buf,len,ndest,1)
c
      go to 10
c
      end
c
c
      subroutine fnode(tp,d,v,nodeid)
      dimension d(3)
c
      go to (1,2,3), nodeid
c
    1 v=-d(1)+d(2)
      return
c
    2 v=d(1)+2*d(2)*d(3))
      return
c
    3 v=d(2)-d(3)
      return
      end
```

LIST OF REFERENCES

1. Roberts, C.E. Jr., *Ordinary Differential Equations*, Prentice-Hall, Inc., 1979.

2. Thomas, G.B. Jr. and Finney, R.L., *Calculus and Analytic Geometry*, Addison-Wesley, 1984.

3. Runge, C., "Ueber die numerishe Auflosung von Differentialgleichungen," *Math. Ann.*, Vol. 46, pp. 167-178.

4. Kutta, W., "Beitrag zur naherungsweisen Integration totaler Differentialgleichungen," *Zeit. Math. Phy.*, Vol. 46, pp. 435-453.

5. Kopal, Z., *Numerical Analysis*, pp. 195-213, Chapman and Hall, 1961.

6. National Aeronautics and Space Administration Technical Report TR R-315, *Low Order Classical Runge-Kutta Formulas With Stepsize Control and Their Application to Some Heat Transfer Problems*, E. Fehlberg, July 1969.

7. Forsythe, G.E., Malcolm, M.A., and Moler, C.B., *Computer Methods for Mathematical Computations*, Prentice-Hall, Inc., 1977.

8. Shampine, L.F., Watts, H.A., and Davenport, S.M., "Solving Nonstiff Ordinary Differential Equations--The State of the Art," *SIAM Review*, Vol. 18, pp. 376-411, 3 July 1976.

9. Hull, T.E., and others, "Comparing Numerical Methods for Ordinary Differential Equations," *SIAM Journal for Numerical Analysis*, Vol. 9.

10. Intel Scientific Computers, *iPSC Technical Description*, S/N 175278-003, September 1986.

BIBLIOGRAPHY

Collatz, L. *The Numerical Treatment of Differential Equations*, Springer-Verlag, 1960.

Conte, S.D. and de Boor, C., *Elementary Numerical Analysis*, McGraw-Hill, 1980.

Gear, C.W., *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, Inc., 1971.

Gerald, C.F. and Wheatley, P.O., *Applied Numerical Analysis*, Addison-Wesley, 1984.

Goldstine, H.H., *A History of Numerical Analysis from the 16th Through the 19th Century*, Springer-Verlag, 1977.

Lapidus, L. and Seinfeld, J.H., *Numerical Solution of Ordinary Differential Equations*, Academic Press, 1971.

Milne, E.W., *Numerical Solution of Differential Equations*, Dover, Inc., 1970.

National Aeronautics and Space Administration Technical Report TR R-287, *Classical Fifth-, Sixth-, Seventh-, and Eighth-Order Runge-Kutta Formulas with Stepsize Control*, E. Fehlberg, October 1968.

Press, W.H., Flannery, B.P., Teukolsky, S.A., and Vetterling, W.T., *Numerical Recipes*, Cambridge University Press, 1986.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Department Chairman, Code 53Fs Department of Mathematics Naval Postgraduate School Monterey, CA 93943-5000	2
4.	Prof. C.E. Roberts, Jr. Mathematics and Computer Science Department Indiana State University Terre Haute, IN 47809	5
4.	Prof. A.L. Schoenstadt, Code 53Zh Department of Mathematics Naval Postgraduate School Monterey, CA 93943-5000	1
5.	LCDR M.L. Mitchell USN, Code 55Mi Department of Operations Research Naval Postgraduate School Monterey, CA 93943-5000	1
6.	MAJ. Rich Adams USAF, Code 52Ad Department of Computer Science Naval Postgraduate School Monterey, CA 93943-5000	1
7.	Mr. Denney R. Cole Intel Scientific Computers 15201 N.W. Greenbrier Parkway Beaverton, OR 97006	1
8.	Captain C.F. Mayo USMC 11027 Stanrich Court Fairfax, VA 22030	6

END

10-87

DTIC