To the Graduate Council:

I am submitting herewith a thesis written by William N. Rust III entitled "Multiprocessor Implementations of a Parallel Implicit Runge-Kutta Code." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Mathematics.

Ohannes Karakashian, Major Professor

We have read this thesis
and recommend its acceptance:

Steven M. Serbin

Lawrence A. Bales

Accepted for the Council:

Vice Provost
and Dean of the Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at The University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature William L. Rust III

Date 11/24/86

MULTIPROCESSOR IMPLEMENTATIONS OF A PARALLEL IMPLICIT RUNGE-KUTTA CODE

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

William N. Rust III

December 1986

ACKNOWLEDGMENTS

The author wishes to thank Jimmie Roberts, Pam Coffey, and Raghu Machiraju of Control Data Corporation for their help and encouragement during the course of this work.

ABSTRACT

A two-stage Implicit Runge-Kutta algorithm is shown to exhibit a high level of parallelism making it an ideal candidate for coding and execution on a two-processor parallel computing system. The algorithm is coded for execution on both a loosely-coupled (CDC Cyberplus) and a tightly-coupled (CRI Cray X-MP) multiprocessor system. The coding and implementation on both systems is described and the performance in terms of speedup is presented. It is shown that the algorithm may be successfully parallel processed on the tightly-coupled system.

TABLE OF CONTENTS

SECTION		PAGE
INTRODUCTION		1
I.	MULTIPROCESSING	2
	Performance	3
	Algorithms	5
	Architectural Considerations	6
II.	A MODEL PROBLEM	8
	Runge-Kutta Methods	8
	The Problem	10
	Theoretical Parallelism	10
	Parallel Tableau	12
	Sequential IRK Implementation	14
	Computational Parallelism	17
III.	CDC CYBERPLUS PARALLEL IRK IMPLEMENTATION	19
	Architecture and System Configuration	19
	System Software	21
	IRK Implementation	23
	Timing	23
	Results	24
IV.	CRI CRAY X-MP IRK IMPLEMENTATION	26
	Architecture	26
	Multitasking Software	26
	IRK Implementation	28
	Timing	28
	Results	29
V.	CONCLUSION	31
REFERENCES		32

PAGE

APPENDIXES 34

A. SEQUENTIAL CODE 35

B. CYBERPLUS CODE 40

C. X-MP CODE 50

VITA 55

LIST OF TABLES

TABLE		PAGE
1.	Maximum Speedup S_{MAX} for Subroutine IRK	18
2.	Cyberplus System Software Subroutines	22
3.	Cyberplus Speedup	25
4.	X-MP Speedup	30

INTRODUCTION

In this thesis we describe the implementation and performance results of an Implicit Runge-Kutta code executed on the CDC Cyberplus and CRI Cray X-MP multiprocessor parallel computers. Our objectives in this work were to (1) gain experience in multiprocessing, (2) determine the difficulty of writing code for multiprocessor parallel computers, and (3) determine speedup factors for Implicit Runge-Kutta algorithms.

We begin in Section I with a general discussion of multiprocessor parallel processing. The purpose of this section is to briefly acquaint the reader with some of the definitions and concepts associated with this area of parallel processing. Our model problem and the Implicit Runge-Kutta solution algorithm are described in Section II. The multiprocessor implementation of the Implicit Runge-Kutta code on the CDC Cyberplus and on the CRI Cray X-MP is described in Sections III and IV, respectively. Concluding remarks are given in the final section.

I. MULTIPROCESSING

Multiprocessing is a type of parallel processing which may be achieved by utilizing more than one processor to execute a single program. Other types of parallel processing include scalar or vector pipelining, array processing, and dataflow. We say that multiprocessing takes place on a multiprocessor; therefore, it is helpful to distinguish between a multiprocessor and a multicomputer.

A multicomputer is a computer system consisting of more than one computer, each capable of executing a stream of instructions independent of the others. A multiprocessor is a multicomputer with additional facilities which enable the computers to communicate so that all can work cooperatively on a single problem. Processor communication is achieved either through a shared memory, through data transfer capabilities, or by both shared memory and data transfer capabilities.

Large mainframe multicomputers have been available for a relatively long time. Multicomputing systems are implemented to increase throughput (the amount of work passing through a computer system) whereas multiprocessing systems are designed to decrease the turnaround time for a single job.

Example multiprocessor parallel processing systems include CRI Cray X-MP, CRI Cray 2, CDC Cyberplus, and ETA Systems ETA^{10}. The Cray X-MP can consist of from one to four vector processors. In its multiprocessor configurations, the X-MP processors communicate

through a shared memory. The Cray 2 has four vector processors which communicate by both shared memory and data transfer. A CDC Cyberplus system consists of from one to 64 scalar processors and a front-end computer, all of which communicate through data transfer capabilities. The ETA^{10} will have from one to eight vector processors with multiprocessor communication available by both shared memory and data transfer.

Important parameters in a multiprocessing computer system are (1) the number of processors, (2) the processor speeds, (3) the local/global memory configuration, and (4) the interprocessor communication and synchronization implementation. Synchronization is necessary in order to insure the proper sequencing of parallel tasks. A terminology has been introduced which provides a convenient way of broadly discussing some of these parameters. A tightly-coupled multiprocessor is one in which communication takes place by way of a shared memory. In a loosely-coupled multiprocessor, communication takes place through a message transfer system. Each processor has its own local memory. An example of a tightly-coupled multiprocessor is the Cray X-MP/4, which is a four processor system. A Cyberplus system is loosely-coupled.

Performance

Let t_p denote the time required to execute a program on a p-processor multiprocessing computing system. The case p = 1 corresponds to sequential execution on a uniprocessor. The speedup S obtained using a p-processor system is defined as the ratio

$$S = \frac{t_1}{t_p} . \qquad (1)$$

Under ideal circumstances one would expect a speedup S = p. Actual speedups are smaller due to the overhead introduced by multiprocessing. This overhead occurs in the form of system-related delays and algorithm-related delays. System-related delays are incurred, for example, during interprocessor communication. Algorithm-related delays are introduced when one or more processors are idle while awaiting results from other processors.

Theoretical estimates for speedup have been proposed. The estimate

$$S = \log_2 p$$

is known as Minsky's Conjecture (Hwang and Briggs, 1984, p. 27). A more illuminating estimate for speedup is made as follows (Buzbee, 1984, pp. 2-3). Define f to be the fraction of work that can be parallel processed. Let t_{si} represent the execution time of systems-related instructions which are necessary to implement multiprocessing. Note that t_{si} may depend upon the number of processors p, the algorithm, the architecture of the machine, and the fraction of work f that may be parallel processed. Then

$$S = \frac{1}{(1-f) + f/p + t_{si}} . \qquad (2)$$

Of particular interest in this estimate for speedup is what it reveals

when f = 1, i.e., the algorithm is completely parallel. In this case,

$$S = \frac{p}{1 + pt_{si}} , \tag{3}$$

which shows that the ideal speedup equal to the number of processors p cannot be obtained.

Algorithms

A parallel algorithm may be defined as a set of tasks which may execute simultaneously to solve a given problem. Tasks communicate to synchronize and to share data.

We may distinguish between two types of synchronization according to characteristics of the system on which it occurs. Implicit synchronization occurs in a loosely-coupled system at a point where a given task is waiting for data produced by another task to be transmitted. Explicit synchronization occurs in a tightly-coupled system at a point where a given task is waiting for data to be produced or updated by another task.

In order to implement a given algorithm in a parallel mode it is necessary to express the algorithm as a set of independent tasks. The identification of tasks which make up the algorithm is referred to as parallel decomposition.

Parallelism in an algorithm may occur at different levels. Low-level parallelism exists at the statement level in a statement such as

$$A = B \cdot C + D \cdot E$$

where the products may be computed in parallel. High-level parallelism exists when groups of statements, such as subroutines, may be executed in parallel. A condition which must be satisfied in order that two subroutines be executed in parallel is known as the Bernstein condition (Hwang and Briggs, 1984, p. 543).

The Bernstein condition is based on two sets of variables for each subroutine S_i. To each subroutine is associated an input set denoted by I_i and an output set denoted by O_i. The input set I_i consists of all memory locations for which the first operation in S_i involving them is a fetch. The output set O_i consists of all memory locations that are stored into by S_i. According to the Bernstein condition,

$$I_1 \cap O_2 = \emptyset$$

and

$$I_2 \cap O_1 = \emptyset$$

must hold if S_1 and S_2 are to be executed in parallel. The Bernstein condition applies more generally to the case where S_1 and S_2 represent statements within a program or subroutine.

Architectural Considerations

The architecture of the machine is the single most important aspect to consider when determining a parallel decomposition for an

algorithm. For example, the cost of communication and data transfer between loosely-coupled processors may be high. Therefore, a loosely-coupled system will likely perform best on a parallel algorithm requiring few interactions among tasks. A common memory eliminates the need for data transmission between processors on a tightly-coupled machine. Thus, an algorithm requiring a high degree of interactions among tasks may perform best on a tightly-coupled system; however, side effects may occur when parallel tasks access and update the same memory location. These observations suggest that a good general strategy for parallel decomposition is to identify tasks in such a way as to balance computation and interprocessor communication so that neither becomes a bottleneck in the overall computation.

II. A MODEL PROBLEM

Our problem consists of approximating the solution to an initial-value problem by way of an Implicit Runge-Kutta (IRK) method. We begin with a general discussion of Runge-Kutta methods and in particular, IRK methods. This is followed by a discussion of the particular initial-value problem with which we are concerned. Next, we specify a subclass of IRK methods which are very suitable for parallel implementation when applied to our model problem. Finally, we describe how the sequential version of our solution routine is implemented in FORTRAN.

Runge-Kutta Methods

Runge-Kutta methods may be characterized by a set of constants which we refer to as the tableau:

$$\begin{array}{c|ccc} a_{11} & \cdots & a_{1q} & \tau_1 \\ \vdots & & \vdots & \\ a_{q1} & \cdots & a_{qq} & \tau_q \\ \hline b_1 & & b_q & \end{array} = \begin{array}{c|c} A & \tau \\ \hline b^T & \end{array} \qquad (4)$$

The integer $q \geq 1$ represents the number of intermediate stages involved in computing the approximation at each time step (this will be clarified below). A Runge-Kutta method is implicit (IRK) if no permutation of rows can reduce A to strict upper or lower triangular form.

Consider the initial-value problem

$$y' = f(t,y), \quad 0<t\leq t^*,$$
$$y(0) = y^0, \tag{5}$$

where y^0 is a given vector in R^n. In order to approximate the solution using tableau (4), we proceed in the following manner. Choose a positive integer n^* and partition the interval $(0,t^*]$ into n^* subintervals of length $k = t^*/n^*$. The length k of each subinterval is referred to as the time step-size. Given y^n, an approximation to $y(t^n)$, we obtain an approximation y^{n+1} to $y(t^{n+1})$ by first computing q intermediate stages

$$y^{n,i} = y^n + k \sum_{j=1}^{q} a_{ij} f(t^{n,j},y^{n,j}), \quad i = 1,\ldots,q \tag{6}$$

where $t^{n,j} = t^n + k\tau_j$, $j=1,\ldots,q$. These values are then combined to yield y^{n+1}:

$$y^{n+1} = y^n + k \sum_{i=1}^{q} b_i f(t^{n,i},y^{n,i}). \tag{7}$$

The methods which we use have the property that the matrix A in the tableau is nonsingular. In case of nonsingular A, Eq. (7) may be written

$$y^{n+1} = (1 - b^T A^{-1} e)y^n + b^T A^{-1} Y, \tag{8}$$

where $b^T = (b_1,\ldots,b_q)$, $e^T = (1,\ldots,1)$, and Y is the R^{nxq} vector $Y^T = ((y^{n,1})^T, (y^{n,2})^T,\ldots,(y^{n,q})^T)$.

The Problem

Consider the initial value problem (5) with $f(t,y) = -Ly$, where $L \in R^{nxn}$ is given by

$$L = (n+1)^2 \begin{bmatrix} 2 & -1 & & 0 \\ -1 & 2 & \ddots & \\ & \ddots & \ddots & -1 \\ 0 & & -1 & 2 \end{bmatrix} .$$

For this choice of f the exact solution is given by

$$y(t) = e^{-tL} y^o .$$

Using a two-stage (q=2) IRK method leads to the following equations for the intermediate stage computations:

$$y^{n,1} = y^n - ka_{11}Ly^{n,1} - ka_{12}Ly^{n,2} \tag{9}$$

$$y^{n,2} = y^n - ka_{21}Ly^{n,1} - ka_{22}Ly^{n,2}. \tag{10}$$

In matrix form, Eqs. (9) and (10) may be expressed

$$\begin{bmatrix} I+ka_{11}L & ka_{12}L \\ ka_{21}L & I+ka_{22}L \end{bmatrix} \begin{bmatrix} y^{n,1} \\ y^{n,2} \end{bmatrix} = \begin{bmatrix} y^n \\ y^n \end{bmatrix} \tag{11}$$

Theoretical Parallelism

Let

$$A = S^{-1} \begin{bmatrix} \lambda_{11} & 0 \\ \theta & \lambda_{22} \end{bmatrix} S$$

be the Jordan form for the tableau matrix A. Here, λ_{11} and λ_{22} are the eigenvalues of A, which we assume are real, and $\theta = 0$ or 1. If $\lambda_{11} \neq \lambda_{22}$, then $\theta = 0$.

With A represented in Jordan form, note that

$$\begin{bmatrix} I+ka_{11}L & ka_{12}L \\ ka_{21}L & I+ka_{22}L \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix} + A \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} kL & 0 \\ 0 & kL \end{bmatrix}$$

$$= \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix} + S^{-1} \begin{bmatrix} \lambda_{11} & 0 \\ \theta & \lambda_{22} \end{bmatrix} S \begin{bmatrix} kL & 0 \\ 0 & kL \end{bmatrix}$$

$$= \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix} + S^{-1} \begin{bmatrix} \lambda_{11}kL & 0 \\ \theta kL & \lambda_{22}kL \end{bmatrix} S$$

$$= S^{-1} \begin{bmatrix} I + \lambda_{11}kL & 0 \\ \theta kL & I + \lambda_{22}kL \end{bmatrix} S ,$$

so that the intermediate stage computation, Eq. (11), becomes

$$S^{-1} \begin{bmatrix} I + \lambda_{11}kL & 0 \\ \theta kL & I + \lambda_{22}kL \end{bmatrix} S \begin{bmatrix} y^{n,1} \\ y^{n,2} \end{bmatrix} = \begin{bmatrix} y^n \\ y^n \end{bmatrix} .$$

Now write

$$\begin{bmatrix} I + \lambda_{11}kL & 0 \\ \theta kL & I + \lambda_{22}kL \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

where

$$\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} s_{11}I & s_{12}I \\ s_{21}I & s_{22}I \end{pmatrix} \begin{pmatrix} y^{n,1} \\ y^{n,2} \end{pmatrix} \tag{12}$$

and

$$\begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} s_{11}I & s_{12}I \\ s_{21}I & s_{22}I \end{pmatrix} \begin{pmatrix} y^{n} \\ y^{n} \end{pmatrix} . \tag{13}$$

Then the intermediate stage computation involves solving the two systems

$$(I + \lambda_{11}kL)u_1 = v_1 \tag{14}$$

$$(I + \lambda_{22}kL)u_2 = v_2 - \theta kLu_1 . \tag{15}$$

It is clear that when $\theta = 0$, Eqs. (14) and (15) may be solved in parallel. We say that a tableau is parallel if $\theta = 0$ in the Jordan form for the tableau matrix A.

Parallel Tableau

The tableau we use comes from the general subclass of tableaus given below (Karakashian, 1986, pp. 77-101). Let

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

$$= \frac{1}{\tau_2 - \tau_1} \begin{pmatrix} \tau_1\tau_2 - \tau_1^2/2 & -\tau_1^2/2 \\ \tau_2^2/2 & -\tau_1\tau_2 + \tau_2^2/2 \end{pmatrix} \tag{16}$$

$$b = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

$$= \frac{1}{\tau_2 - \tau_1} \begin{pmatrix} \tau_2 - 1/2 \\ -\tau_1 + 1/2 \end{pmatrix} \tag{17}$$

and

$$\tau = \begin{pmatrix} \tau_1 \\ \tau_2 \end{pmatrix}$$

$$= \begin{pmatrix} \lambda_{11} + \lambda_{22} + \sqrt{\lambda_{11}^2 + \lambda_{22}^2} \\ \lambda_{11} + \lambda_{22} - \sqrt{\lambda_{11}^2 + \lambda_{22}^2} \end{pmatrix} . \tag{18}$$

A parallel tableau is obtained by choosing λ_{11} and λ_{22} so that

$$\lambda_{11} > 1/2,$$

$$\lambda_{22} > 1/2,$$

and

$$\lambda_{11}\lambda_{22} - (1/2)(\lambda_{11} + \lambda_{22}) + 1/6 = 0. \tag{19}$$

In this case a Jordan form for the tableau matrix A has the form

$$A = S^{-1} \begin{pmatrix} \lambda_{11} & 0 \\ 0 & \lambda_{22} \end{pmatrix} S,$$

where

$$S = \begin{bmatrix} 1 & (\lambda_{11} - a_{11})/a_{21} \\ (\lambda_{22} - a_{22})/a_{12} & 1 \end{bmatrix} . \qquad (20)$$

Sequential IRK Implementation

The computational flow for the sequential IRK implementation is shown in Figure 1. A few observations are in order concerning the indicated calculations.

First note that the coefficient matrices in Eqs. (14) and (15) are positive definite. Thus we may use the Cholesky method to obtain the factorizations shown in Steps 4 and 5. The main loop consists of Steps 9 through 13. Note that the transformation given in Eq. (13) is applied outside of the main loop. Equations (14) and (15) are solved within the main loop. Thus, Eq. (8) must be modified before it can be used to calculate the solution at the next step. Premultiplying each term of Eq. (8) by S we obtain

$$S\begin{bmatrix} y^{n+1} \\ y^{n+1} \end{bmatrix} = (1 - b^T A^{-1} e) S \begin{bmatrix} y^n \\ y^n \end{bmatrix} + S \begin{bmatrix} b^T A^{-1} Y \\ b^T A^{-1} Y \end{bmatrix} .$$

Note that we may replace Y with $S^{-1}SY$ to obtain

$$S\begin{bmatrix} y^{n+1} \\ y^{n+1} \end{bmatrix} = (1 - b^T A^{-1} e) S \begin{bmatrix} y^n \\ y^n \end{bmatrix} + S \begin{bmatrix} b_1 & b_2 \\ b_1 & b_2 \end{bmatrix} A^{-1} S^{-1} S \begin{bmatrix} y^{n,1} \\ y^{n,2} \end{bmatrix} . \qquad (21)$$

The terms $(1 - b^T A^{-1} e)$ and $(S[b^T\ b^T]^T A^{-1} S^{-1})$ are constants and, as shown in Figure 1, are calculated outside of the main loop in Step 6.

Upon completion of the main loop, the reverse transformation is applied to obtain the desired solution.

STEP 1. Specify input parameters.

n (dimension), t^*, n^* (number of subintervals), λ_{11}

STEP 2. Load initial value for y.

$y^0 = (y_1^0, y_2^0, \ldots, y_n^0)$

STEP 3. Calculate program parameters.

k (step-size), $\lambda_{22}, \tau_1, \tau_2, b_1, b_2, A^{-1}, S, S^{-1}$

STEP 4. Factor $(I + \lambda_{11} kL)$.

STEP 5. Factor $(I + \lambda_{22} kL)$.

STEP 6. Calculate loop constants.

$c = (1 - b^T A^{-1} e)$

$$M = \begin{pmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{pmatrix} = S \begin{pmatrix} b_1 & b_2 \\ b_1 & b_2 \end{pmatrix} A^{-1} S^{-1}$$

STEP 7. Perform transformation.

$$\begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} s_{11} I & s_{12} I \\ s_{21} I & s_{22} I \end{pmatrix} \begin{pmatrix} y^0 \\ y^0 \end{pmatrix}$$

STEP 8. Initialize for main loop.

$i = 1$

STEP 9. Solve system 1.

$(I + \lambda_{11} kL) u_1 = v_1$

STEP 10. Solve system 2.

$(I + \lambda_{22} kL) u_2 = v_2$

Figure 1. IRK computational sequence.

STEP 11. Calculate solution at next step.

$$\begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = C \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} + M \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}$$

STEP 12. Increment loop counter.

set i = i + 1

STEP 13. If $ki \leqslant t^*$, go to STEP 9.

STEP 14. Perform transformation.

$$\begin{pmatrix} y^{t^*} \\ y^{t^*} \end{pmatrix} = S^{-1} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$$

STEP 15. Output solution.

Figure 1 (continued)

The complete FORTRAN listing of the sequential code is contained in Appendix A. The code consists of a driver routine (DRSEQ) and four subroutines (IRK, PGMPAR, CHLSKY, and SOLVE). The driver routine sets up input parameters and passes them to subroutine IRK which directs the solution calculation. The computationally intense parts of subroutine IRK have been isolated in subroutines CHLSKY and SOLVE. CHLSKY obtains the factorizations shown in Steps 4 and 5 of Figure 1. SOLVE solves the systems shown in Steps 9 and 10 of Figure 1.

Recall that high-level parallelism exists in a code when two or more subroutines may be executed in parallel. In case of subroutine IRK, high-level parallelism exists in the two calls to subroutine CHLSKY and in the two calls within the main loop to subroutine SOLVE. By modularizing the code in this way, we obtain an ideal code structure for parallel processing on a two-processor multiprocessor system.

Computational Parallelism

An experiment was conducted to estimate the maximum speedup attainable by multiprocessing subroutine IRK on a two-processor system. Note that subroutine IRK calls subroutine CHLSKY twice. Subroutine SOLVE is called twice for each iteration of the main loop. Therefore, the parallel execution time of subroutine IRK, assuming no overhead due to multiprocessor communication and synchronization, is

$$t_{PARALLEL} = t_{IRK} - t_{CHLSKY} - n^{*} \cdot t_{SOLVE} .$$

Using Eq. (1) for speedup, we find that the maximum speedup attainable by multiprocessing subroutine IRK is

$$S_{MAX} = \frac{t_{IRK}}{t_{PARALLEL}} .$$

Execution times were obtained for subroutines CHLSKY and SOLVE after which subroutine IRK was timed. Values of S_{MAX} which were calculated for some representative problems are shown in Table 1.

Table 1. Maximum Speedup S_{MAX} for Subroutine IRK

	Number of Steps		
Dimension	100	500	1000
1000	1.9799	1.9833	1.9842
800	1.9790	1.9826	1.9830
600	1.9775	1.9820	1.9825
400	1.9759	1.9798	1.9803
200	1.9888	1.9723	1.9732
50	1.9409	1.9312	1.9312

Recall from Eq. (3) that the maximum theoretical speedup that may be attained by using a two-processor multiprocessor system is bounded above by two. Factors which prevent attaining the maximum speedup are the efficiency of the multiprocessor communication and synchronization implementation and the degree of parallelism inherent in the code. As Table 1 indicates, subroutine IRK has a high level of inherent parallelism. We may conclude from this that multiprocessing overhead will be the overriding barrier to achieving optimal speedup.

III. CDC CYBERPLUS PARALLEL IRK IMPLEMENTATION

Architecture and System Configuration

A Cyberplus computing system consists of a CDC Cyber 170 Series 800 front-end computer running the NOS 2 operating system, from one to 64 Cyberplus processors, the interface between the front-end computer and the Cyberplus processors, and the related software. The Cyber 170 front-end computer, henceforth referred to as the Cyber, provides the input-output facility and initiates and terminates execution of code on the Cyberplus processors. The Cyberplus processors may be organized into four groups of up to 16 processors per group. Processors within a group are connected by way of a system ring, which in turn is connected to a Cyber System Ring Interface unit (SRI) which provides the interface between the Cyber and the Cyberplus processors on each ring. Software for the Cyberplus consists of system routines which provide a communication link between the Cyber and the Cyberplus, a machine instruction language compiler, a FORTRAN compiler, a simulator, and various utilities.

The Cyberplus system we used is located at the University of Georgia's Advanced Computational Methods Center. This system has a Cyber 845 for the front-end computer and two system rings with an integer processor on one ring and two floating-point processors on the other.

The Cyber computer consists of a central processor, central memory, peripheral processors, data channels, and a system console.

The peripheral processors have access to central memory and do all input and output so that the central processor is relieved of this task. The data channels link peripheral equipment to the peripheral processors through a matrix. The system console monitors and effects system activity.

A possible limiting factor in the Cyberplus configuration is the manner in which the SRI is connected to the Cyber. The SRI is connected through a standard data channel. This implies that data being transferred between the Cyber and Cyberplus must pass through a peripheral processor, a data channel, and the SRI before reaching its destination. The communication delay incurred during data transmission may degrade the performance of the system for certain applications.

The Cyberplus processors have a 21 nsec clock cycle. There are 15 segmented functional units which are connected via a crossbar switch which allows functional units to send and receive data every clock cycle. Four additional functional units come with the floating-point option. The functional units are capable of a certain level of simultaneous operation thus enabling the processor to exploit low-level parallelism (statement level); however, the current version of the FORTRAN compiler does not effectively take advantage of this capability. Access to this feature of the machine is available only through the machine instruction language for the computer.

Base memory on the Cyberplus processors consists of 256K 64 bit words organized into four banks. An additional 256K of memory is optional and increases the number of memory banks from four to eight. There is no shared memory on the Cyberplus system.

Important differences exist between the Cyber and the Cyberplus processors in the way data are represented in these machines. As mentioned earlier, the Cyberplus processor uses a 64 bit word. Integer data, for example, are represented in two's complement form. The front-end Cyber uses a 60 bit word in one's complement form to represent integer data. Similarly, real data are represented differently with exponent fields of differing lengths and biases. The differences in the way data are represented in these machines lead to overhead which is incurred during data conversion.

System Software

The System Software provides a communication link between the Cyber and the Cyberplus processors on each ring. The software consists of subroutines which are callable from FORTRAN. These subroutines allow a user to load and execute a Cyberplus FORTRAN program and to transfer data back and forth between the Cyber and the Cyberplus processors during program execution.

Table 2 lists the System Software subroutines which were used to implement the IRK code on the Cyberplus processors. The Z and X routines are called from FORTRAN executing on the Cyber whereas the HLOC and HSMSG routines are called from Cyberplus FORTRAN. The X routines are used to load and execute a Cyberplus FORTRAN program or subroutine. The remaining routines shown in Table 2 are used for communication and data transfer.

Table 2. Cyberplus System Software Subroutines

Subroutine	Function
XINIT	Initialize Cyberplus FORTRAN environment (presets internal tables)
XPARM	Define parameter locations
XLOAD	Load Cyberplus subroutine (no parameters loaded)
XGO	Execute Cyberplus subroutine (transfers data then initiates execution)
XEND	Perform termination processing of Cyberplus subroutine (transfer parameters to Cyber)
ZIC2H	Convert integer data from Cyber format to Cyberplus format
ZIH2C	Convert integer data from Cyberplus format to Cyber format
ZRHPM	Transfer data from a Cyberplus to the Cyber
ZWHPM	Transfer data from the Cyber to a Cyberplus
ZRDMSG	Read oldest message sent from a Cyberplus
HLOC	Determine address of a variable in a Cyberplus
HSMSG	Send a message from a Cyberplus to the Cyber

IRK Implementation

The basic approach taken in the Cyberplus IRK implementation was to group the subroutines so that the degree of interaction within groups was greater than the degree of interaction between groups. This approach was taken in order to minimize communication and data transfer between processors.

Each processor was loaded with copies of subroutines PGMPAR, CHLSKY, and SOLVE. Separate versions of subroutine IRK were loaded in each processor. For processor 1, IRK was modified to deal only with the computations associated with Eq. (14). As such, the call to CHLSKY to factor the coefficient matrix of Eq. (15) was removed as was the call to SOLVE to solve system (15). Similar modifications were made to create the complementary version of IRK for execution on processor 2. Also, the appropriate code was added to each version to effect data transmission and synchronization through the front-end.

A FORTRAN listing of the front-end parallel driver routine and the IRK versions for each Cyberplus processor is contained in Appendix B. A listing of the front-end sequential driver routine is also included. The sequential code consisted of the original versions of subroutines IRK, PGMPAR, CHLSKY, and SOLVE.

Timing

In order to obtain accurate turnaround times for the Cyberplus code, a routine was written to read the real-time clock on the front-end Cyber. The standard FORTRAN call to the SECOND routine could not be used since this routine returns accumulated cpu time for code

executing on the front-end. When the front-end code is in a rollout state and not executing, such as during data transmission, cpu time is not accumulated; however, the time required for data transmission is important in a parallel environment since it represents overhead.

Results

The performance in speedup for subroutine IRK is indicated in Table 3. Results are given as a function of dimension n and number of steps n^*. The number of steps is equivalent to the number of iterations in the main loop.

As indicated in Table 3, no speedup was attained by multiprocessing subroutine IRK on the Cyberplus. In all cases, the sequential version of the code ran faster than the parallel version. The overhead incurred during data transmission severely degraded the performance of the system when executing the parallel version of the code.

Table 3. Cyberplus Speedup

	Number of Steps									
Dimension	100	200	300	400	500	600	700	800	900	1000
1000	0.266	0.221	0.196	0.185	0.190	0.165	0.163	0.159	0.158	0.152
950	0.223	0.223	0.210	0.201	0.170	0.164	0.166	0.159	0.156	0.154
900	0.269	0.229	0.220	0.178	0.169	0.169	0.161	0.155	0.154	0.150
850	0.265	0.227	0.210	0.179	0.165	0.169	0.159	0.156	0.153	0.151
800	0.306	0.227	0.200	0.184	0.170	0.169	0.153	0.155	0.157	0.152
750	0.286	0.227	0.187	0.177	0.166	0.161	0.157	0.150	0.148	0.144
700	0.268	0.227	0.182	0.181	0.168	0.161	0.155	0.151	0.145	0.144
650	0.285	0.225	0.182	0.172	0.157	0.151	0.149	0.144	0.140	0.140
600	0.276	0.217	0.238	0.178	0.166	0.153	0.151	0.146	0.143	0.138
550	0.274	0.203	0.195	0.173	0.154	0.147	0.144	0.139	0.134	0.135
500	0.273	0.219	0.190	0.176	0.158	0.159	0.146	0.138	0.136	0.130
450	0.299	0.250	0.197	0.198	0.158	0.144	0.143	0.132	0.132	0.124
400	0.263	0.225	0.186	0.191	0.150	0.140	0.130	0.130	0.126	0.120
350	0.321	0.223	0.187	0.168	0.146	0.145	0.163	0.136	0.125	0.121
300	0.296	0.228	0.183	0.163	0.155	0.135	0.133	0.123	0.117	0.111
250	0.302	0.212	0.196	0.190	0.184	0.137	0.126	0.120	0.115	0.111
200	0.333	0.222	0.188	0.158	0.139	0.125	0.123	0.114	0.113	0.102
150	0.381	0.229	0.193	0.189	0.134	0.127	0.118	0.108	0.108	0.104
100	0.372	0.218	0.168	0.147	0.137	0.120	0.116	0.098	0.095	0.088
50	0.323	0.268	0.192	0.167	0.167	0.120	0.114	0.100	0.093	0.086

IV. CRI CRAY X-MP IRK IMPLEMENTATION

Architecture

A Cray X-MP computing system may consist of either one, two, or four vector processors. The processors have a 9.5 nsec clock cycle. The processors share a central memory which consists of up to 16 million 64 bit words organized into 16 or 32 banks depending on memory size. An I/O subsystem functions as the interface between the X-MP and front-end computers.

The X-MP system we used is located at the Pittsburgh Supercomputing Center and consists of four processors with eight million words of shared memory. The system runs the Cray Operating System (COS) and has two clustered DEC VAX 8650 computers for front-ends.

A processor intercommunication section of the system is used for applying multiple processors to a single job. The section consists of groups of shared registers which allow processors to exchange signals indicating that processing should begin, should wait, or has been completed (Larson, 1984, p. 62). These registers are allocated by the operating system to a multiprocessing job.

Multitasking Software

The Cray Multitasking Library is used to gain access to the processor intercommunication section of the system. The library consists of subroutines which are callable from FORTRAN.

Before discussing the library subroutines, it is necessary to understand the notion of task as it applies to the Multitasking Library. The operating system handles tasks as the basic unit of work. The main program is a task. In general, a task is a unit of computation such as a program or subroutine that can be scheduled for execution. Multiprocessing, or multitasking as it is referred to here, occurs when a program or subroutine creates additional tasks which may then be scheduled for simultaneous execution. The current implementation of multitasking only allows for multitasking of subroutines.

Task creation takes place via the call to subroutine TSKSTART. The arguments for TSKSTART include a task control array and the name of the subroutine for which the separate task is being created. If the multitasked subroutine has arguments, then these too are included in the argument list to TSKSTART. The task control array is used by the operating system to identify and schedule tasks.

The program or subroutine creating a separate task calls subroutine TSKWAIT in order to halt execution until the separate task has completed. The argument to TSKWAIT is the task control array used in the corresponding call to TSKSTART. The TSKWAIT subroutine serves as a synchronization mechanism.

An alternative way to synchronize the execution of tasks is to use the EVPOST and EVWAIT subroutines. In order to use this method, an event variable is declared in the main task with the EVASN subroutine. A task may then use the event variable as an argument in

the EVPOST subroutine to signal to other tasks the completion of an operation. Other tasks wait for the operation to finish by calling subroutine EVWAIT with the event variable as an argument.

IRK Implementation

The basic approach taken in the X-MP IRK implementation was to evenly distribute the parallel portions of the computation among processors. The first call to subroutine CHLSKY was replaced with a call to TSKSTART to create a separate task. Once this was created and scheduled for execution, the second call to CHLSKY was initiated. Synchronization took place with a call to TSKWAIT just before the main loop. In the main loop, the first call to subroutine SOLVE was removed and a separate task created with TSKSTART. Upon return from the second call to SOLVE, TSKWAIT was called to synchronize.

A FORTRAN listing of the X-MP parallel IRK code is contained in Appendix C.

Timing

The standard FORTRAN call to the SECOND routine was used to obtain sequential code execution times. In order to obtain accurate parallel execution times in a batch environment, it was necessary to schedule the runs to execute in a dedicated mode. The times obtained were the wall clock times registered by the COS IRTC routine.

Results

The performance in speedup for subroutine IRK is indicated in Table 4. Results are given as a function of dimension n and number of steps n^*.

As indicated in Table 4, consistent speedups were attained for dimension n larger than 100. In addition, the speedups improved with increasing n. A comparison of Table 4 with the experimental results given in Table 1 (p. 18) indicate that the overhead in using the Multitasking Library routines is minimal.

Table 4. X-MP Speedup

	Number of Steps		
Dimension	50	100	200
5000	1.854	1.888	1.885
2000	1.810	1.597	1.855
1500	1.697	1.782	1.823
1000	1.909	1.901	1.863
900	1.894	1.894	1.866
800	1.866	1.876	1.856
700	1.863	1.869	1.842
600	1.846	1.854	1.827
500	1.802	1.831	1.790
400	1.791	1.772	1.671
300	1.741	1.704	1.701
200	1.589	1.596	1.565
100	0.944	1.016	1.134
50	1.080	1.043	1.031
40	0.969	0.953	0.967
30	0.840	0.814	0.832
20	0.611	0.538	0.643
10	0.215	0.244	0.326

V. CONCLUSION

Based on the speedup attained by multiprocessing subroutine IRK on the X-MP, it is clear that a two-stage Implicit Runge-Kutta algorithm can successfully be parallel processed. Moreover, the ease with which the subroutine was implemented offers bright prospects for extension to four-stage methods.

The data communication and synchronization overhead on the Cyberplus system was no match for the speed of a single Cyberplus executing subroutine IRK. With sequential execution times as much as 10 times faster than parallel execution times, it is clear that a computation of this size is not appropriate for the system. A larger, more intense problem requiring more computation between synchronization points would better test the Cyberplus parallel performance.

REFERENCES

REFERENCES

Buzbee, B. L.,"The Efficiency of Parallel Processing," Los Alamos National Laboratory Technical Report, LA-UR-84-2097, Los Alamos, New Mexico, 1984.

Control Data Corporation, "CDC Cyberplus FORTRAN Reference Manual," Publication No. 60461880, St. Paul, Minnesota, 1985.

Control Data Corporation, "Cyberplus Parallel Processing System: System Software and Utilities Reference Manual," Publication No. 60461680 C, St. Paul, Minnesota, 1985.

Cray Research Inc., "FORTRAN (CFT) Reference Manual," Publication No. SN-0009, Mendota Heights, Minnesota, 1986.

Cray Research Inc., "Multitasking User Guide," Publication No. SN-0222, Mendota Heights, Minnesota, 1986.

ETA Systems, "ETA^{10} System Overview," Publication No. 1006, St. Paul, Minnesota, 1986.

Hwang, K. and F. A. Briggs, Computer Architecture and Parallel Processing, McGraw-Hill, New York, 1984.

Karakashian, O. A., "On Runge-Kutta Methods for Parabolic Problems with Time Dependent Coefficients," Math Comp., 47 (July 1986), pp. 77-101.

Larson, J. L., "Multitasking on the Cray X-MP/2 Multiprocessor," IEEE Computer, Vol. 17, No. 7, July 1984, pp. 62-69.

Stewart, G. W., Introduction to Matrix Computations, Academic Press, New York, 1973.

APPENDIXES

APPENDIX A

SEQUENTIAL CODE

```
      PROGRAM DRSEQ(INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT)
      PARAMETER(LN=1000)
      DIMENSION Y(LN)
C.....SPECIFY INPUT PARAMETERS
      N = 100
      TSTAR = 1.0
      NSTAR = 100
      IPO = 1
      RLAM11 = 0.75
      STEPS = NSTAR
      T1 = SECOND()
C.....LOAD INITIAL VALUE FOR Y
      DO 110 I = 1, N
        Y(I) = 1.0
110   CONTINUE
C.....CALL SOLUTION ROUTINE
      CALL IRK(N,TSTAR,STEPS,RLAM11,Y)
      T2 = SECOND()
      T = T2 - T1
      WRITE(6,97)T
97    FORMAT(' ',F12.5,' SECONDS')
C.....OUTPUT SOLUTION
      WRITE(6,98)STEPS
98    FORMAT(' YI, I =',F5.0)
      WRITE(6,99)(Y(I),I=1,N,IPO)
99    FORMAT(5(2X,F12.5))
      STOP
      END

      SUBROUTINE IRK(N,TSTAR,STEPS,RLAM11,Y)
      PARAMETER(LN=1000)
      DIMENSION D1(LN),S1(LN-1),D2(LN),S2(LN-1)
      DIMENSION Y(LN),V1(LN),V2(LN),U1(LN),U2(LN)
      DINENSION Z(LN),W1(LN),W2(LN),RM(2,2)
      COMMON/PAR/RK,RLAM22,AI(2,2),T1,T2,B1,B2
     2 S(2,2),SI(2,2)
C.....CALCULATE PROGRAM PARAMETERS
      CALL PGMPAR(TSTAR,STEPS,RLAM11)
C.....FACTOR (I + RLAM11*RK*L)
      CALL CHLSKY(RLAM11,RK,D1,S1,N)
C.....FACTOR (I + RLAM22*RK*L)
      CALL CHLSKY(RLAM22,RK,D2,S2,N)
C.....CALCULATE LOOP CONSTANTS
      C = 1.0 - B1*(AI(1,1) + AI(1,2)) - B2*(AI(2,1))
     2    + AI(2,2))
      F1 = S(1,1) + S(1,2)
      F2 = S(2,1) + S(2,2)
      G1 = B1*(AI(1,1)*SI(1,1) + AI(1,2)*SI(2,1))
```

```
     2     + B2*(AI(2,1)*SI(1,1) + AI(2,2)*SI(2,1))
      G2 = B1*(AI(1,1)*SI(1,2) + AI(1,2)*SI(2,2))
     2     + B2*(AI(2,1)*SI(1,2) + AI(2,2)*SI(2,2))
      RM(1,1) = F1 * G1
      RM(1,2) = F1 * G2
      RM(2,1) = F2 * G1
      RM(2,2) = F2 * G2
C.....PERFORM TRANSFORMATION
      DO 10 I = 1, N
        V1(I) = F1 * Y(I)
        V2(I) = F2 * Y(I)
10    CONTINUE
C.....INITIALIZE FOR MAIN LOOP
      STEP = 1.0
C.....MAIN LOOP
20    CONTINUE
C.....SOLVE SYSTEM 1.  STORE SOLUTION IN U1.
      CALL SOLVE(D1,S1,V1,U1,Z,N)
C.....SOLVE SYSTEM 2.  STORE SOLUTION IN U2.
      CALL SOLVE(D2,S2,V2,U2,Z,N)
C.....CALCULATE SOLUTION AT NEXT STEP
      DO 30 I = 1, N
        W1(I) = RM(1,1) * U1(I) + RM(1,2) * U2(I)
        W2(I) = RM(2,1) * U1(I) + RM(2,2) * U2(I)
        V1(I) = C * V1(I) + W1(I)
        V2(I) = C * V2(I) + W2(I)
30    CONTINUE
C.....INCREMENT STEP AND TEST
      STEP = STEP + 1.0
      IF(STEP.GT.STEPS)GO TO 40
      GO TO 20
40    CONTINUE
C.....PERFORM TRANSFORMATION
      DO 50 I = 1, N
        Y(I) = SI(1,1)*V1(I) + SI(1,2)*V2(I)
50    CONTINUE
      RETURN
      END

      SUBROUTINE PGMPAR(TSTAR,STEPS,RLAM11)
      COMMON/PAR/RK,RLAM22,AI(2,2),T1,T2,B1,B2
     2 S(2,2),SI(2,2)
C.....CALCULATE STEP-SIZE
      RK = TSTAR / STEPS
C.....SOLVE FOR LAMBDA 22
      RLAM22 = (3.0*RLAM11 - 1.0) / (6.0*RLAM11 - 3.0)
C.....CALCULATE THE TABLEAU VECTORS TAU AND B,
C.....THE TABLEAU MATRIX A, AND A INVERSE
```

```
      T1 = RLAM11 + RLAM22 + SQRT(RLAM11*RLAM11
     2      + RLAM22*RLAM22)
      T2 = 2.0*(RLAM11 + RLAM22) - T1
      T = T2 - T1
      B1 = (T2 - 0.5) / T
      B2 = (0.5 - T1) / T
      A11 = (T1*T2 - 0.5*T1*T1) / T
      A12 = (-0.5*T1*T1) / T
      A21 = (0.5*T2*T2) / T
      A22 = (0.5*T2*T2 - T1*T2) / T
      DEN = A11*A22 - A12*A21
      AI(1,1) = A22 / DEN
      AI(1,2) = -1.0*A12 / DEN
      AI(2,1) = -1.0*A21 / DEN
      AI(2,2) = A11 / DEN
C.....CALCULATE THE MATRICES S AND S INVERSE IN
C.....THE JORDAN FORM FOR THE TABLEAU MATRIX A
      S(1,1) = 1.0
      S(1,2) = (RLAM11 - A11) / A21
      S(2,1) = (RLAM22 - A22) / A12
      S(2,2) = 1.0
      DEN = S(1,1)*S(2,2) - S(1,2)*S(2,1)
      SI(1,1) = S(2,2) / DEN
      SI(1,2) = -1.0*S(1,2) / DEN
      SI(2,1) = -1.0*S(2,1) / DEN
      SI(2,2) = S(1,1) / DEN
      RETURN
      END

      SUBROUTINE CHLSKY(RLAM,RK,D,S,N)
      PARAMETER(LN=1000)
      DIMENSION D(LN),S(LN-1)
C
C     CHLSKY CALCULATES THE CHOLESKY FACTORIZATION OF
C     (I + RLAM*RK*L), WHERE I AND L ARE NXN MATRICES,
C     DIA(L) = 2.0 * N**2, AND SUBDIA(L) = SUPDIA(L) =
C     -1.0 * N**2.  THE FACTORIZATION IS STORED IN VECTORS
C     D(N) AND S(N-1), WHERE D(N) CONTAINS THE DIAGONAL
C     OF EACH FACTOR AND S(N-1) CONTAINS THE SUBDIAGONAL
C     OF THE LEFTHAND FACTOR (SUPDIAGONAL OF THE
C     RIGHTHAND FACTOR).
C
      RN1SQ = (N+1) * (N+1)
      B = 1.0 + 2.0 * RLAM * RK * RN1SQ
      C = -1.0 * RLAM * RK * RN1SQ
C.....CHOLESKY FACTORIZATION
      D(1) = SQRT(B)
      DO 10 I = 1, N-1
```

```
          S(I) = C / D(I)
          TERM = B - S(I) * S(I)
          D(I+1) = SQRT(TERM)
10      CONTINUE
        RETURN
        END

        SUBROUTINE SOLVE(D,S,Y,X,Z,N)
        PARAMETER(LN=1000)
        DIMENSION D(LN),S(LN-1),Y(LN),X(LN),Z(LN)
C
C       SOLVE USES THE CHOLESKY FACTORIZATION
C       OF (I + RLAM*RK*L) TO SOLVE THE SYSTEM
C       (I + RLAM*RK*L) X = Y.
C
C.....FORWARD SOLVE
        Z(1) = Y(1) / D(1)
        DO 10 I = 2, N
          Z(I) = (Y(I) - S(I-1) * Z(I-1)) / D(I)
10      CONTINUE
C.....BACKWARD SOLVE
        X(N) = Z(N) / D(N)
        DO 20 J = 1, N-1
          I = N - J
          X(I) = (Z(I) - S(I) * X(I+1)) / D(I)
20      CONTINUE
        RETURN
        END
```

APPENDIX B

CYBERPLUS CODE

```
C
C
C       CYBERPLUS FRONT-END PARALLEL DRIVER ROUTINE
C
C
        PROGRAM DRPAR(INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT)
        PARAMETER(LN=1000)
        INTEGER HPMLEN
        PARAMETER(HPMLEN=(LN*16/15)+2)
        INTEGER LU1HPP0,LU2HPP0,LU1HPP1,LU2HPP1
        INTEGER G0,HPMG0,MSG
        DIMENSION Y(LN),U1(HPMLEN),U2(HPMLEN)
        INTEGER HPS,HPP0,HPP1
        PARAMETER(HPS=1,HPP0=0,HPP1=1)
C.....SPECIFY INPUT PARAMETERS
        N = 100
        TSTAR = 1.0
        NSTAR = 100
        IP0 = 1
        RLAM11 = 0.75
        STEPS = NSTAR
        CALL RTC(ITIME1)
C*************************************************************
C.....SPECIFY INITIAL VALUE FOR Y

        DO 110 I = 1, N
          Y(I) = 1.0
110     CONTINUE

C.....SYNCHRONIZATION PARAMETERS

        G0  = 1
        CALL ZIC2H(G0,HPMG0,1,ICALL)

C.....INITIALIZE CYBERPLUS FORTRAN ENVIRONMENT

        CALL XINIT(ICALL,HPS,0,0,0)

C.....DEFINE PARAMETER LOCATIONS FOR HPP0

        CALL XPARM(HPS,HPP0,1,N,1,ICALL)
        CALL XPARM(HPS,HPP0,2,TSTAR,1,ICALL)
        CALL XPARM(HPS,HPP0,3,STEPS,1,ICALL)
        CALL XPARM(HPS,HPP0,4,RLAM11,1,ICALL)
        CALL XPARM(HPS,HPP0,5,Y,N,ICALL)

C.....DEFINE PARAMETER LOCATIONS FOR HPP1
```

```
      CALL XPARM(HPS,HPP1,1,N,1,ICALL)
      CALL XPARM(HPS,HPP1,2,TSTAR,1,ICALL)
      CALL XPARM(HPS,HPP1,3,STEPS,1,ICALL)
      CALL XPARM(HPS,HPP1,4,RLAM11,1,ICALL)
      CALL XPARM(HPS,HPP1,5,Y,N,ICALL)

C.....LOAD ROUTINES IN HPP0 AND HPP1

      CALL XLOAD(HPS,HPP0,'OBJ0',ICALL)
      CALL XLOAD(HPS,HPP1,'OBJ1',ICALL)

C.....START EXECUTION OF ROUTINES AND RETURN CONTROL TO FE

      CALL XGO(HPS,HPP0,0,ICALL)
      CALL XGO(HPS,HPP1,0,ICALL)

C.....READ ADDRESSES OF U1 AND U2 IN EACH HPP

      CALL ZRDMSG(HPS,IHPP,MSG,1,ICALL)
      LG0HPP0 = SHIFT(MSG,-16)
      CALL ZRDMSG(HPS,IHPP,MSG,1,ICALL)
      LU1HPP0 = SHIFT(MSG,-16)
      CALL ZRDMSG(HPS,IHPP,MSG,1,ICALL)
      LU2HPP0 = SHIFT(MSG,-16)
      CALL ZRDMSG(HPS,IHPP,MSG,1,ICALL)
      LG0HPP1 = SHIFT(MSG,-16)
      CALL ZRDMSG(HPS,IHPP,MSG,1,ICALL)
      LU1HPP1 = SHIFT(MSG,-16)
      CALL ZRDMSG(HPS,IHPP,MSG,1,ICALL)
      LU2HPP1 = SHIFT(MSG,-16)

C.....SYNCHRONIZATION - TELL HPPS TO CONTINUE

      CALL ZWHPM(HPS,HPP0,LG0HPP0,1,HPMG0,0,ICALL)
      CALL ZWHPM(HPS,HPP1,LG0HPP1,1,HPMG0,0,ICALL)


C.....TRANSFER LOOP

      DO 130 S = 1, STEPS
        DO 120 IWAIT = 0, 1
          CALL ZRDMSG(HPS,IHPP,MSG,1,ICALL)
          IF(IHPP.EQ.0)THEN
            CALL ZRHPM(HPS,IHPP,LU1HPP0,N+1,U1(1),
     2                 IWAIT,ICALL)
          ELSE
            CALL ZRHPM(HPS,IHPP,LU2HPP1,N+1,U2(1),
     2                 IWAIT,ICALL)
```

```
            ENDIF
120       CONTINUE
            CALL ZWHPM(HPS,HPP0,LU2HPP0,N+1,U2(1),0,ICALL)
            CALL ZWHPM(HPS,HPP1,LU1HPP1,N+1,U1(1),1,ICALL)
130     CONTINUE

C.....END PROCESSING OF HPP ROUTINES

        CALL XEND(HPS,HPP0,1,ISTAT,ICALL)
        CALL XEND(HPS,HPP1,1,ISTAT,ICALL)

C*****************************************************************
        CALL RTC(ITIME2)
        ITIME = ITIME2 - ITIME1
        TPAR = ITIME / 1000.0
        WRITE(6,97)TPAR
97      FORMAT(' TPAR =',F10.3)
C.....OUTPUT SOLUTION
        WRITE(6,98)STEPS
98      FORMAT(' YI, I =',F5.0)
        WRITE(6,99)(Y(I),I=1,N,IP0)
99      FORMAT(5(2X,F12.5))
        STOP
        END
            IDENT  RTC
...
ASSEMBLY CODE TO GET REAL TIME CLOCK
IN MILLISECONDS GOES HERE
...
            END
```

```
C
C
C      CYBERPLUS SUBROUTINE IRK - PROCESSOR 1 VERSION
C
C
       SUBROUTINE IRK0(N,TSTAR,STEPS,RLAM11,Y)
       PARAMETER(LN=1000)
       INTEGER G0,ADDRG0(3),MSG(3)
       INTEGER ADDRU1(3),ADDRU2(3)
       DIMENSION D1(LN),S1(LN-1)
       DIMENSION Y(LN),V1(LN),V2(LN),U1(LN),U2(LN)
       DINENSION Z(LN),W1(LN),W2(LN),RM(2,2)
       COMMON/PAR/RK,RLAM22,AI(2,2),T1,T2,B1,B2
      2 S(2,2),SI(2,2)
       U1(N+1) = 1.0
       G0 = 0
C.....SEND ADDRESS OF G0 TO FRONT END
       CALL HLOC(G0,ADDRG0)
       ADDRG0(3) = 0
       CALL HSMSG(ADDRG0)
C.....SEND ADDRESS OF U1 TO FRONT END
       CALL HLOC(U1,ADDRU1)
       ADDRU1(3) = 0
       CALL HSMSG(ADDRU1)
C.....SEND ADDRESS OF U2 TO FRONT END
       CALL HLOC(U2,ADDRU2)
       ADDRU2(3) = 0
       CALL HSMSG(ADDRU2)
C.....CALCULATE PROGRAM PARAMETERS
       CALL PGMPAR(TSTAR,STEPS,RLAM11)
C.....FACTOR (I + RLAM11*RK*L)
       CALL CHLSKY(RLAM11,RK,D1,S1,N)
C.....CALCULATE LOOP CONSTANTS
       C = 1.0 - B1*(AI(1,1) + AI(1,2)) - B2*(AI(2,1))
      2    + AI(2,2))
       F1 = S(1,1) + S(1,2)
       F2 = S(2,1) + S(2,2)
       G1 = B1*(AI(1,1)*SI(1,1) + AI(1,2)*SI(2,1))
      2     + B2*(AI(2,1)*SI(1,1) + AI(2,2)*SI(2,1))
       G2 = B1*(AI(1,1)*SI(1,2) + AI(1,2)*SI(2,2))
      2     + B2*(AI(2,1)*SI(1,2) + AI(2,2)*SI(2,2))
       RM(1,1) = F1 * G1
       RM(1,2) = F1 * G2
       RM(2,1) = F2 * G1
       RM(2,2) = F2 * G2
C.....PERFORM TRANSFORMATION
       DO 10 I = 1, N
         V1(I) = F1 * Y(I)
```

```
        V2(I) = F2 * Y(I)
10    CONTINUE
C.....INITIALIZE FOR MAIN LOOP
      STEP = 1.0
C.....MAIN LOOP
20    CONTINUE
C.....SOLVE SYSTEM 1. STORE SOLUTION IN U1.
      CALL SOLVE(D1,S1,V1,U1,Z,N)
C.....SYNCHRONIZE
30    CONTINUE
      IF(GO.EQ.0)GO TO 30
C.....GET U2 FROM HPP1
      U2(N+1) = -1.0
      CALL HSMSG(MSG)
40    CONTINUE
      IF(U2(N+1).EQ.-1.0)GO TO 40
C.....CALCULATE SOLUTION AT NEXT STEP
      DO 50 I = 1, N
        W1(I) = RM(1,1) * U1(I) + RM(1,2) * U2(I)
        W2(I) = RM(2,1) * U1(I) + RM(2,2) * U2(I)
50    CONTINUE
      DO 60 I = 1, N
        V1(I) = C * V1(I) + W1(I)
        V2(I) = C * V2(I) + W2(I)
60    CONTINUE
C.....INCREMENT STEP AND TEST
      STEP = STEP + 1.0
      IF(STEP.GT.STEPS)GO TO 70
      GO TO 20
70    CONTINUE
C.....PERFORM TRANSFORMATION
      DO 80 I = 1, N
        Y(I) = SI(1,1)*V1(I) + SI(1,2)*V2(I)
80    CONTINUE
      RETURN
      END
```

```
C
C
C      CYBERPLUS SUBROUTINE IRK - PROCESSOR 2 VERSION
C
C
       SUBROUTINE IRK1(N,TSTAR,STEPS,RLAM11,Y)
       PARAMETER(LN=1000)
       INTEGER GO,ADDRGO(3),MSG(3)
       INTEGER ADDRU1(3),ADDRU2(3)
       DIMENSION D2(LN),S2(LN-1)
       DIMENSION Y(LN),V1(LN),V2(LN),U1(LN),U2(LN)
       DINENSION Z(LN),W1(LN),W2(LN),RM(2,2)
       COMMON/PAR/RK,RLAM22,AI(2,2),T1,T2,B1,B2
      2 S(2,2),SI(2,2)
       U2(N+1) = 1.0
       GO = 0
C.....SEND ADDRESS OF GO TO FRONT END
       CALL HLOC(GO,ADDRGO)
       ADDRGO(3) = 0
       CALL HSMSG(ADDRGO)
C.....SEND ADDRESS OF U1 TO FRONT END
       CALL HLOC(U1,ADDRU1)
       ADDRU1(3) = 0
       CALL HSMSG(ADDRU1)
C.....SEND ADDRESS OF U2 TO FRONT END
       CALL HLOC(U2,ADDRU2)
       ADDRU2(3) = 0
       CALL HSMSG(ADDRU2)
C.....CALCULATE PROGRAM PARAMETERS
       CALL PGMPAR(TSTAR,STEPS,RLAM11)
C.....FACTOR (I + RLAM22*RK*L)
       CALL CHLSKY(RLAM22,RK,D2,S2,N)
C.....CALCULATE LOOP CONSTANTS
       C = 1.0 - B1*(AI(1,1) + AI(1,2)) - B2*(AI(2,1))
      2    + AI(2,2))
       F1 = S(1,1) + S(1,2)
       F2 = S(2,1) + S(2,2)
       G1 = B1*(AI(1,1)*SI(1,1) + AI(1,2)*SI(2,1))
      2     + B2*(AI(2,1)*SI(1,1) + AI(2,2)*SI(2,1))
       G2 = B1*(AI(1,1)*SI(1,2) + AI(1,2)*SI(2,2))
      2     + B2*(AI(2,1)*SI(1,2) + AI(2,2)*SI(2,2))
       RM(1,1) = F1 * G1
       RM(1,2) = F1 * G2
       RM(2,1) = F2 * G1
       RM(2,2) = F2 * G2
C.....PERFORM TRANSFORMATION
       DO 10 I = 1, N
         V1(I) = F1 * Y(I)
```

```
        V2(I) = F2 * Y(I)
10      CONTINUE
        STEP = 1.0
C.....MAIN LOOP
20      CONTINUE
C.....SOLVE SYSTEM 2. STORE SOLUTION IN U2.
        CALL SOLVE(D2,S2,V2,U2,Z,N)
C.....SYNCHRONIZE
30      CONTINUE
        IF(GO.EQ.0)GO TO 30
C.....GET U1 FROM HPPO
        U1(N+1) = -1.0
        CALL HSMSG(MSG)
40      CONTINUE
        IF(U1(N+1).EQ.-1.0)GO TO 40
C.....CALCULATE SOLUTION AT NEXT STEP
        DO 50 I = 1, N
          W1(I) = RM(1,1) * U1(I) + RM(1,2) * U2(I)
          W2(I) = RM(2,1) * U1(I) + RM(2,2) * U2(I)
50      CONTINUE
        DO 60 I = 1, N
          V1(I) = C * V1(I) + W1(I)
          V2(I) = C * V2(I) + W2(I)
60      CONTINUE
C.....INCREMENT STEP AND TEST
        STEP = STEP + 1.0
        IF(STEP.GT.STEPS)GO TO 70
        GO TO 20
70      CONTINUE
C.....PERFORM TRANSFORMATION
        DO 80 I = 1, N
          Y(I) = SI(1,1)*V1(I) + SI(1,2)*V2(I)
80      CONTINUE
        RETURN
        END
```

```
C
C
C      CYBERPLUS FRONT-END SEQUENTIAL DRIVER ROUTINE
C
C
       PROGRAM DRSEQ(INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT)
       PARAMETER(LN=1000)
       DIMENSION Y(LN)
       INTEGER HPS,HPP0,HPP1
       PARAMETER(HPS=1,HPP0=0,HPP1=1)
C.....SPECIFY INPUT PARAMETERS
       N = 100
       TSTAR = 1.0
       NSTAR = 100
       IP0 = 1
       RLAM11 = 0.75
       STEPS = NSTAR
       CALL RTC(ITIME1)
C*************************************************************
C.....SPECIFY INITIAL VALUE FOR Y

       DO 110 I = 1, N
         Y(I) = 1.0
110    CONTINUE

C.....INITIALIZE CYBERPLUS FORTRAN ENVIRONMENT

       CALL XINIT(ICALL,HPS,0,0,0)

C.....DEFINE PARAMETER LOCATIONS FOR HPP0

       CALL XPARM(HPS,HPP0,1,N,1,ICALL)
       CALL XPARM(HPS,HPP0,2,TSTAR,1,ICALL)
       CALL XPARM(HPS,HPP0,3,STEPS,1,ICALL)
       CALL XPARM(HPS,HPP0,4,RLAM11,1,ICALL)
       CALL XPARM(HPS,HPP0,5,Y,N,ICALL)

C.....LOAD ROUTINE IN HPP0

       CALL XLOAD(HPS,HPP0,'OBJSEQ',ICALL)

C.....START EXECUTION OF ROUTINE AND WAIT FOR COMPLETION

       CALL XGO(HPS,HPP0,1,ICALL)

C*************************************************************
       CALL RTC(ITIME2)
       ITIME = ITIME2 - ITIME1
```

```
      TSEQ = ITIME / 1000.0
      WRITE(6,97)TSEQ
97    FORMAT(' TSEQ =',F10.3)
C.....OUTPUT SOLUTION
      WRITE(6,98)STEPS
98    FORMAT(' YI, I =',F10.3)
      WRITE(6,99)(Y(I),I=1,N,IPO)
99    FORMAT(5(2X,F12.5))

      STOP
      END
          IDENT  RTC
...
ASSEMBLY CODE TO GET REAL TIME CLOCK
IN MILLISECONDS GOES HERE
...
          END
```

APPENDIX C

X-MP CODE

CRAY X-MP PARALLEL IRK CODE

```
JOB,JN=MT,US=.
ACCOUNT,AC=,PW=.
MULTI.
CFT,ALLOC=STACK,L=0.
LDR,MAP=OFF.
/EOF
      PROGRAM DRIVER
      PARAMETER(LN=1000)
      PARAMETER(CLOCK=9.5E-9)
      DIMENSION Y(LN)
C.....SPECIFY INPUT PARAMETERS
      N = 1000
      TSTAR = 1.0
      STEPS = 1000
      IPO = 50
      RLAM11 = 0.75
      I1 = IRTC()
C.....LOAD INITIAL VALUE FOR Y
      DO 110 I = 1, N
        Y(I) = 1.0
110   CONTINUE
C.....CALL SOLUTION ROUTINE
      CALL IRK(N,TSTAR,STEPS,RLAM11,Y)
      I2 = IRTC()
      ITIME = I2 - I1
      T = FLOAT(ITIME) * CLOCK
      WRITE(6,97)T
97    FORMAT(' TPAR = ',F12.5)
C.....OUTPUT SOLUTION
      WRITE(6,98)STEPS
98    FORMAT(' STEPS = ',F5.0)
      WRITE(6,99)(Y(I),I=1,N,IPO)
99    FORMAT(5(F16.9))
      STOP
      END

      SUBROUTINE IRK(N,TSTAR,STEPS,RLAM11,Y)
      PARAMETER(LN=1000)
CRAY
      INTEGER TASKARY(2)
      EXTERNAL CHLSKY,SOLVE
CRAY
      DIMENSION D1(LN),S1(LN-1),D2(LN),S2(LN-1)
      DIMENSION Y(LN),V1(LN),V2(LN),U1(LN),U2(LN)
      DIMENSION Z1(LN),Z2(LN),W1(LN),W2(LN),RM(2,2)
```

```
      COMMON/PAR/RK,RLAM22,AI(2,2),T1,T2,B1,B2,
     2 S(2,2),SI(2,2)
C.....CALCULATE PROGRAM PARAMETERS
      CALL PGMPAR(TSTAR,STEPS,RLAM11)
CRAY
      TASKARY(1) = 2
      NCPU = 4
      CALL TSKTUNE('MAXCPU',NCPU)
C.....FACTOR (I + RLAM11*RK*L)
      CALL TSKSTART(TASKARY,CHLSKY,RLAM11,RK,D1,S1,N)
CRAY
C.....FACTOR (I + RLAM22*RK*L)
      CALL CHLSKY(RLAM22,RK,D2,S2,N)
C.....CALCULATE LOOP CONSTANTS
      C = 1.0 - B1*(AI(1,1) + AI(1,2)) - B2*(AI(2,1)
     2    + AI(2,2))
      F1 = S(1,1) + S(1,2)
      F2 = S(2,1) + S(2,2)
      G1 = B1*(AI(1,1)*SI(1,1) + AI(1,2)*SI(2,1))
     2     + B2*(AI(2,1)*SI(1,1) + AI(2,2)*SI(2,1))
      G2 = B1*(AI(1,1)*SI(1,2) + AI(1,2)*SI(2,2))
     2     + B2*(AI(2,1)*SI(1,2) + AI(2,2)*SI(2,2))
      RM(1,1) = F1 * G1
      RM(1,2) = F1 * G2
      RM(2,1) = F2 * G1
      RM(2,2) = F2 * G2
C.....PERFORM TRANSFORMATION
      DO 10 I = 1, N
        V1(I) = F1 * Y(I)
        V2(I) = F2 * Y(I)
10    CONTINUE
C.....INITIALIZE FOR MAIN LOOP
      STEP = 1.0
CRAY
      CALL TSKWAIT(TASKARY)
CRAY
C.....MAIN LOOP
20    CONTINUE
C.....SOLVE SYSTEM 1.  STORE SOLUTION IN U1.
CRAY
      CALL TSKSTART(TASKARY,SOLVE,D1,S1,V1,U1,Z1,N)
CRAY
C.....SOLVE SYSTEM 2.  STORE SOLUTION IN U2.
      CALL SOLVE(D2,S2,V2,U2,Z2,N)
CRAY
      CALL TSKWAIT(TASKARY)
CRAY
C.....CALCULATE SOLUTION AT NEXT STEP
```

```
      DO 30 I = 1, N
        W1(I) = RM(1,1) * U1(I) + RM(1,2) * U2(I)
        W2(I) = RM(2,1) * U1(I) + RM(2,2) * U2(I)
        V1(I) = C * V1(I) + W1(I)
        V2(I) = C * V2(I) + W2(I)
30    CONTINUE
C.....INCREMENT STEP AND TEST
      STEP = STEP + 1.0
      IF(STEP.GT.STEPS)GO TO 40
      GO TO 20
40    CONTINUE
C.....PERFORM TRANSFORMATION
      DO 50 I = 1, N
        Y(I) = SI(1,1)*V1(I) + SI(1,2)*V2(I)
50    CONTINUE
      RETURN
      END

      SUBROUTINE PGMPAR(TSTAR,STEPS,RLAM11)
      COMMON/PAR/RK,RLAM22,AI(2,2),T1,T2,B1,B2,
     2 S(2,2),SI(2,2)
C.....CALCULATE STEP-SIZE
      RK = TSTAR / STEPS
C.....SOLVE FOR LAMBDA 22
      RLAM22 = (3.0*RLAM11 - 1.0) / (6.0*RLAM11 - 3.0)
C.....CALCULATE THE TABLEAU VECTORS TAU AND B,
C.....THE TABLEAU MATRIX A, AND A INVERSE
      T1 = RLAM11 + RLAM22 + SQRT(RLAM11*RLAM11
     2     + RLAM22*RLAM22)
      T2 = 2.0*(RLAM11 + RLAM22) - T1
      T = T2 - T1
      B1 = (T2 - 0.5) / T
      B2 = (0.5 - T1) / T
      A11 = (T1*T2 - 0.5*T1*T1) / T
      A12 = (-0.5*T1*T1) / T
      A21 = (0.5*T2*T2) / T
      A22 = (0.5*T2*T2 - T1*T2) / T
      DEN = A11*A22 - A12*A21
      AI(1,1) = A22 / DEN
      AI(1,2) = -1.0*A12 / DEN
      AI(2,1) = -1.0*A21 / DEN
      AI(2,2) = A11 / DEN
C.....CALCULATE THE MATRICES S AND S INVERSE IN
C.....THE JORDAN FORM FOR THE TABLEAU MATRIX A
      S(1,1) = 1.0
      S(1,2) = (RLAM11 - A11) / A21
      S(2,1) = (RLAM22 - A22) / A12
      S(2,2) = 1.0
```

```
      DEN = S(1,1)*S(2,2) - S(1,2)*S(2,1)
      SI(1,1) = S(2,2) / DEN
      SI(1,2) = -1.0*S(1,2) / DEN
      SI(2,1) = -1.0*S(2,1) / DEN
      SI(2,2) = S(1,1) / DEN
      RETURN
      END

      SUBROUTINE CHLSKY(RLAM,RK,D,S,N)
      PARAMETER(LN=1000)
      DIMENSION D(LN),S(LN-1)
      RN1SQ = (N+1) * (N+1)
      B = 1.0 + 2.0 * RLAM * RK * RN1SQ
      C = -1.0 * RLAM * RK * RN1SQ
C.....CHOLESKY FACTORIZATION
      D(1) = SQRT(B)
      DO 10 I = 1, N-1
        S(I) = C / D(I)
        TERM = B - S(I) * S(I)
        D(I+1) = SQRT(TERM)
10    CONTINUE
      RETURN
      END

      SUBROUTINE SOLVE(D,S,Y,X,Z,N)
      PARAMETER(LN=1000)
      DIMENSION D(LN),S(LN-1),Y(LN),X(LN),Z(LN)
C.....FORWARD SOLVE
      Z(1) = Y(1) / D(1)
      DO 10 I = 2, N
        Z(I) = (Y(I) - S(I-1) * Z(I-1)) / D(I)
10    CONTINUE
C.....BACKWARD SOLVE
      X(N) = Z(N) / D(N)
      DO 20 J = 1, N-1
        I = N - J
        X(I) = (Z(I) - S(I) * X(I+1)) / D(I)
20    CONTINUE
      RETURN
      END
/EOF
```

VITA

William N. Rust III was born in Manchester, Tennessee, on July 10, 1956. He was graduated from Bearden High School in Knoxville in June, 1974. After a four-year tour of duty with the United States Air Force, he entered The University of Tennessee, Knoxville, and in June, 1983, he received his Bachelor of Arts degree in Mathematics.

He was employed with Arvin-Calspan as a computer scientist for one year after which, in September, 1984, he accepted a teaching assistantship at The University of Tennessee, Knoxville. He received his Master of Science in Mathematics in December, 1986.

The author has been employed with Control Data Corporation since December, 1985.