To the Graduate Council:

I am submitting herewith a thesis written by Kenneth William Fister entitled "Parallel Implementations of Certain Algorithms for the Linear Inhomogeneous Heat and Wave Equations." I have examined the final copy of the 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.

Steven M. Serbin

Steven M. Serbin, Major Professor

We have read this thesis
and recommend its acceptance:

A. Samuel Jordan

Charles H. Conley

Accepted for the Council:

Carolyn Minkel

Associate Vice Chancellor
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 the 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 Kenneth William Fiske

Date December 1, 1994.

Parallel Implementations of Certain Algorithms for the Linear Inhomogeneous Heat and Wave Equations

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Kenneth William Fister

December 1994

DEDICATION

To my wonderful wife, Renee, whose constant prayers, support, and encouragement helped me through this degree, even when finishing seemed like a long way from happening.

ACKNOWLEDGEMENTS

I would like to thank Dr. Steve Serbin for his support and guidance throughout this thesis. Also, I would like to thank my other committee members, Dr. Sam Jordan and Dr. Chuck Collins, for their comments and suggestions. Finally, I would like to thank both my wife's family and my family for all their prayers and encouragement.

ABSTRACT

In this thesis we look at certain algorithms for the inhomogeneous heat and wave equations, with the focus being the implementation of these algorithms in PVM (Parallel Virtual Machine). PVM creates, via connection of workstations, a parallel environment. These algorithms are created in a special way to ensure that a phenomenon known as order reduction will not occur; our numerical experiments confirm this. The parallelism in these algorithms is obtained by the use of partial fraction expansion of appropriate rational functions. Numerical evidence is furnished to show that the algorithms perform essentially the same in both serial and parallel mode.

TABLE OF CONTENTS

0 Introduction 1

1 The Heat Equation 3
1.1 A Fourth Order Scheme As A Special Case Of A General Second Order Scheme 3
1.2 Parallelizing the Heat Equation 7

2 The Wave Equation 10
2.1 A Fourth Order Scheme As A Special Case Of A General Second Order Scheme 10
2.2 A Sixth Order Scheme As A Special Case Of A General Fourth Order Scheme . 14
2.3 Obtaining A Sixth Order Parallel Algorithm 24

3 Conclusion 27

LIST OF REFERENCES 29

APPENDIX 30

A PVM Basics 31
A-1 Running PVM 31
A-2 The PVM Commands Used In This Thesis 32

B Programs Used in This Thesis 33
`par24-seq-quad.for` 34
`par24-master.for` 41
`par24-worker.for` 45
`hyp24-seq-quad.for` 52
`hyp46-seq-quad.for` 59
`hyp46-pf-quad.for` 69
`hyp-master.for` 78
`hyp-worker.for` 83

VITA 90

LIST OF TABLES

1.1.1 Results from the fourth order algorithm for parabolic problems 6

1.1.2 Results when the number of both the spatial and time subintervals are incremented in algorithm for parabolic problems 6

1.2.1 Results showing PVM version of algorithm, **par24-master.for** with **par24-worker.for**, gives the same error as serial version, **par24-seq-doub.for** . . . 9

1.2.2 Results comparing the time it took the serial version, **par24-seq-doub.for**, and PVM version, **par24-master.for** with **par24-worker.for**, to complete the specified problem . 9

2.1.1 Results from program **hyp24-seq-quad.for** showing that this algorithm is generally second order but is fourth order for the specified values of β and θ 15

2.1.2 Results from program **hyp24-seq-quad.for** showing similar results as the number of space and time subintervals are both incremented 16

2.2.1 Results from program **hyp46-seq-quad.for** showing that this algorithm is generally fourth order, but it is sixth order for the specified values of β_i and τ_i 22

2.2.2 Results from program **hyp46-seq-quad.for** showing similar results as the number of time and space subintervals are both incremented 23

2.3.1 Results showing that the partial fractions version in serial mode, **hyp46-pf-doub.for**, and the corresponding parallel version, **hyp-master.for** with **hyp-worker.for**, give comparable error values to the original serial version, **hyp46-seq-doub.for** 26

2.3.2 Results showing the time comparisons for finishing certain problems by the serial version, **hyp46-seq-doub.for**; partial fraction version in serial mode, **hyp46-pf-doub.for**; and the corresponding PVM version, **hyp-master.for** with **hyp-worker.for** 26

Chapter 0

Introduction

In this thesis, we will look at certain algorithms for numerically solving the linear inhomogeneous heat and wave equations. In the process we will implement these algorithms initially in the standard serial mode. Some of these algorithms have parts which can be solved independently; hence, they will be suitable to run in parallel. We will convert these algorithms into code which we will run in a parallel environment. We will, also, look at how these algorithms numerically compare to the theoretical results when performed in floating point arithmetic. Also, these algorithms were designed to avoid order reduction, a phenomenon which appears in many methods.

Using classical techniques to solve the inhomogeneous heat and wave equations, one would use finite elements or finite difference methods on the spatial variables to discretize the equation into a system of ordinary differential equations. Then one may attempt to solve this system using any of the numerical methods for solving ordinary differential equations. For some of these methods, like certain Runga-Kutta methods, the order that the error is reduced as the mesh is refined is not as high as one expects. Verwer [8] shows several numerical examples that illustrate this phenomenon, called order reduction.

For the inhomogeneous heat equation Brenner, Crouzeix, and Thomée [1] develop a scheme with certain conditions in which order reduction does not occur. Serbin [7] developed this method further using rational functions which could then be broken down using partial fractions. This scheme is one which is suitable for implementing in parallel. Serbin [7] implemented one version of this technique but at the time had no immediate access to a parallel machine and hence the parallelism was simulated. We have further tested this method in serial mode and then converted the version to run in a parallel environment.

In order to test the parallel version we have used a software package called PVM which stands for Parallel Virtual Machine. Basically, PVM works by connecting networked machines,

using each machine as if it were a processor in some larger entity. This enables the user to work in a parallel environment without having access to a single high-performance parallel computer. In this case, we have used for our PVM nodes machines on the UTK Mathematics Department SUN network. PVM function calls are inserted into the different programs which are usually one program to be executed in parallel and another which sends and collects the data from the parallel execution. These function calls allow the different programs and processors to communicate with each other, thus acting as a parallel virtual machine.

Dougalis and Makridakis [3] proposed a similar method for the linear inhomogeneous wave equation; their proposed method is similar in nature to Brenner, Crouzeix, and Thomée's method for the heat equation. We will take this method and, as in Serbin [7], will develop a new implementation using rational functions and partial fractions. This method will be suitable for a parallel environment. Once again we will test both versions in standard serial mode and the latter in parallel.

Chapter 1

The Heat Equation

1.1 A Fourth Order Scheme As A Special Case Of A General Second Order Scheme

The scheme used here is introduced in Brenner, Crouzeix, and Thomée [1] and developed further by Serbin [7]. We will not discuss the development of the scheme for this parabolic problem, but in a subsequent chapter we will sketch the development of a similar method for hyperbolic problems. We first take the inhomogeneous heat equation and develop an algorithm which is suitable for being parallelized. The basic problem is to find the function $u(x,t)$ which satisfies

$$\begin{aligned} u_t - \Delta u &= f \quad \text{in } \Omega \times [0,\infty), \\ u &= 0 \quad \text{on } \partial\Omega \times [0,\infty), \\ u(\cdot,0) &= 0 \quad \text{in } \Omega. \end{aligned} \tag{1.1.1}$$

Ω is a bounded domain in R^d with smooth boundary and Δ is the usual Laplacian. For our problem, we will take $\Omega = [0,1]$ and $t \in [0,1]$. Our ultimate approximation scheme will employ a temporal stepsize k and spatial stepsize h, thus involving the grid points $t_n = nk$ and $x_j = jh$. First, by standard techniques we semidiscretize the problem to produce the system

$$\frac{d}{dt}u_h(t) - \Delta_h u_h(t) = (P_0 f)(t), \tag{1.1.2}$$

where $u_h(t)$ is an element of an appropriate finite dimensional subspace S_h of functions of x, $P_0 f$ is the L_2 projection of f onto S_h, and Δ_h, the semidiscrete Laplacian, maps $S_h \mapsto S_h$ to provide an approximation to Δ. We then denote by $\{\phi_i\}_{i=1,\ldots,N}$ a basis for S_h; for ease of exposition, we will assume that this constitutes a nodal basis, in the sense that $\phi_i(x_j) = \delta_{ij}$.

Proceeding toward a full discretization, the usual integrating factor approach leads to

$$u_h(t_{n+1}) = e^{k\Delta_h}u_h(t_n) + k\int_0^1 e^{k(1-\tau)\Delta_h}P_0 f(t_n + k\tau)d\tau. \tag{1.1.3}$$

Now, express $u_h(t) = \sum_{i=1}^{N} U_i(t)\phi_i$, so that

$$\sum_{i=1}^{N} U_i(t_{n+1})\phi_i(x_j) = e^{k\Delta_h} \sum_{i=1}^{N} U_i(t_n)\phi_i(x_j) + k \int_0^1 e^{k(1-\tau)\Delta_h} P_0 f(t_n + k\tau)(x_j)d\tau, \tag{1.1.4}$$

which results in

$$U_j(t_{n+1}) = e^{k\Delta_h} U_j(t_n) + k \int_0^1 e^{k(1-\tau)\Delta_h} P_0 f(t_n + k\tau)(x_j)d\tau. \tag{1.1.5}$$

Finally, we may then approximate $U_j(t_{n+1})$ by U_j^{n+1} according to

$$U_j^{n+1} = r(k\Delta_h)U_j^n + k \sum_{i=1}^{\mu} q_i(k\Delta_h) P_0 f(t_n + \tau_i k)(x_j), \qquad j = 1, \ldots, N, \tag{1.1.6}$$

which collectively becomes the vector system

$$U^{n+1} = r(k\Delta_h)U^n + k \sum_{i=1}^{\mu} q_i(k\Delta_h) P_0 f(t_n + \tau_i k), \qquad n = 0, 1, 2, \ldots, \tag{1.1.7}$$

with $U^0 = 0$. We note that r and q_i are appropriate rational functions to be selected later, that it is customary to suppress the x_j, and that we have now associated the operator Δ_h with its matrix representation with respect to the chosen basis and similarly $P_0 f(\xi)$ with a vector. Finally, we note that with appropriate choice of S_h, this scheme may be interpreted in the context of finite differences, as we shall do below. To summarize, when we compute the vector U^n, $n = 0, 1, 2, \ldots$, we are producing the approximation $u(x_j, t_n) \approx U_j^n$.

Using partial fractions we express

$$r(\lambda) = \sum_{j=0}^{\mu} \frac{a_j}{1 - \gamma_j \lambda} \tag{1.1.8a}$$

and

$$q_i(\lambda) = \sum_{j=1}^{\mu} \frac{\delta_j^{(i)}}{1 - \gamma_j \lambda} \tag{1.1.8b}$$

where $r(\lambda)$ approximates e^{λ}, and the functions $q_i(\lambda)$ enter in the approximation of certain integrals. The γ_j will be selected below as distinct reciprocal poles, and the remaining coefficients are then determined from the scheme.

Thus substituting (1.1.3) into (1.1.2), we see that $u(x,t)$ can be approximated using the following algorithm. First solve for $W^{n,j}$ in

$$(I - \gamma_j k\Delta_h)W^{n,j} = a_j U^n + k\sum_{i=1}^{\mu}\delta_j^{(i)} P_0 f(t_n + \tau_i k), \qquad j = 1,\ldots,\mu \tag{1.1.9a}$$

and then calculate

$$U^{n+1} = a_0 U^n + \sum_{j=1}^{\mu} W^{n,j}, \qquad n = 0,1,2,\ldots \tag{1.1.9b}$$

where $\tau_1 = 0, \tau_2 = 0.5$, and $\tau_3 = 1$. These are by no means the only choice of quadrature points, but they are the ones reported on by Serbin [7]. Moreover, the choice of the γ_j's is explained by Serbin [7] and has to do with an idea called the "real-pole sandwich" (see also, Norsett and Wanner [5]). Choosing $\gamma_1 = 1.06$, $\gamma_2 = 1.07$, and $\gamma_3 = 1.075890333521764$ gives for the a_j's:

$a_0 = -.6304261005497924$, $a_1 = 1,454.094874359267$, $a_2 = -3,794.673554929712$, and $a_3 = 2,342.209106670995$. For the above γ_j's Serbin [7] determines the following $\delta_j^{(i)}$'s:

$\delta_1^{(1)} = 3567.278607850113$, $\delta_2^{(1)} = -9621.288547664747$, $\delta_3^{(1)} = 6054.176606481301$,

$\delta_1^{(2)} = -7319.518083718724$, $\delta_2^{(2)} = 19811.01919385797$, $\delta_3^{(2)} = -12490.83444347257$,

$\delta_1^{(3)} = 5293.580042689434$, $\delta_2^{(3)} = -14250.03134996801$, and $\delta_3^{(3)} = 8956.617973945243$. We use a uniform grid which has $N+1$ spatial subintervals and $K+1$ time subintervals; thus, $h = \frac{1}{N+1}$ and $k = \frac{1}{K+1}$. We use the second order finite difference scheme for spatial discretization and get the system $U_t + AU = F(t)$ so that

$$-\Delta_h = A = \frac{1}{h^2}\begin{pmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & -1 & \ddots & \vdots \\ 0 & -1 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{pmatrix}$$

and $P_0 f = F$. We will test this with $u(x,t) = t^\alpha x(1-x)$ with $\alpha = 3$. We make this choice so that the error is controlled only by the time stepsize. This exact solution gives $F_i(t) = t^{\alpha-1}[\alpha x_i(1 - x_i) + 2t]$. We look at the error at $t = 1$, where we use the norm $\| \varepsilon \|_\infty \equiv \max_{1\le i\le N} \mid u(x_i,1) - U_i^K \mid$. What we shall see is that with the above choices of γ_i, we obtain a fourth order method.

The above algorithm can be set up in parallel by solving on different processors the μ different equations obtained in (1.1.9a). Then the results can be gathered from the different processors to calculate U^{n+1} in (1.1.9b).

To verify that this is fourth order we run the program, **par24-seq-quad.for**, and vary the time step size while holding the spatial step size constant, (see Table 1.1.1). We discussed earlier how the error, ε, is determined. To compute the order, here and throughout this thesis, we use the method given in Serbin [7]. We assume that in our test problems the error satisfies approximately a relationship of the form $\varepsilon = ck^p$, where c does not depend on k (Verwer [8]). Taking adjacent entries in the table we estimate the order by

$$\widehat{p} = \ln\left(\frac{\varepsilon_i}{\varepsilon_j}\right) / \ln\left(\frac{k_i}{k_j}\right). \tag{1.1.10}$$

We see that fourth order is approached, and our results coincide with those reported in Serbin [7] for the partial fraction algorithm. Table 1.1.2 shows we get similar results when the number of both the spatial and time subintervals are incremented, so order reduction is not apparent.

Table 1.1.1: Results from the fourth order algorithm for parabolic problems (i.e., from program **par24-seq-quad.for**)

h	k	ERROR	$\widehat{p}$
$\frac{1}{10}$	$\frac{1}{10}$	4.4739E-05	
$\frac{1}{10}$	$\frac{1}{20}$	5.5055E-06	3.0226
$\frac{1}{10}$	$\frac{1}{40}$	5.3230E-07	3.3706
$\frac{1}{10}$	$\frac{1}{80}$	4.2199E-08	3.6570
$\frac{1}{10}$	$\frac{1}{160}$	2.9236E-09	3.8514
$\frac{1}{10}$	$\frac{1}{320}$	1.9167E-10	3.9310

Table 1.1.2: Results when the number of both the spatial and time subintervals are incremented in algorithm for parabolic problems (i.e., from program **par24-seq-quad.for**)

h	k	ERROR	$\widehat{p}$
$\frac{1}{10}$	$\frac{1}{10}$	4.4739E-05	
$\frac{1}{20}$	$\frac{1}{20}$	5.5154E-06	3.0200
$\frac{1}{40}$	$\frac{1}{40}$	5.3488E-07	3.3662
$\frac{1}{80}$	$\frac{1}{80}$	4.2473E-08	3.6546
$\frac{1}{160}$	$\frac{1}{160}$	2.9395E-09	3.8529

1.2 Parallelizing the Heat Equation.

In order to implement a program in PVM, the program first needs to be divided into two parts: the master and the worker. The worker performs the calculations that can be done in parallel; in our case this would be the μ forward and back solves to get $W^{n,j}$. The master program would perform the rest of the calculations including gathering the $W^{n,j}$'s and calculating (1.1.9b). When implementing the program in parallel using PVM there are a few commands that will need to be used. The appendix and the **PVM 3 User's Guide and Reference Manual**, [6], should be referenced to better understand these commands. The PVM commands used in the master program would be **pvmfmytid**, **pvmfspawn**, **pvmfinitsend**, **pvmfpack**, **pvmfsend**, **pvmfrecv**, **pvmfunpack**, and **pvmfexit**. In the worker program the PVM commands used are **pvmfmytid**, **pvmfparent**, **pvmfrecv**, **pvmfunpack**, **pvmfinitsend**, **pvmfpack**, **pvmfsend**, and **pvmfexit**.

To compare the results between the serial and parallel implementations we are actually comparing the clock time, not C.P.U. time, from the start of the program to the end. This will give a fairer comparison, because we want to look at how fast the parallel version is compared to the serial. Also, when setting up the PVM virtual machine we need to make sure all machines are of the same speed; we accomplish this by picking machines that have the same architecture. Otherwise, we will have some processors sitting by while the slowest among them are struggling to finish each particular subtask. For the heat equation, since we are using $\mu = 3$ we will use IPC's, so we can use 4 in our virtual machine. However, after understanding how PVM works, we actually only need 3 machines for our choice of $\mu = 3$. For, the machine that is running the master program can also be assigned a subtask; if not, it will be waiting around to gather the results from each of the subtasks. Hence, we can set up a virtual machine consisting of faster machines like the Sparc 10 Model 40's, of which we have access to only three.

We can implement algorithm (1.1.9b) by factoring $(I - \gamma_j k \Delta_h)$ on the left hand side of (1.1.9a). We first used LU decomposition, but after further investigation, we switched to LL^t decomposition to exploit symmetry and positive definiteness. Initially, we also set up the program

so that the decomposition occurred each time the loop was run. After careful examination we see that the μ different decompositions can be performed before the loop is entered. Then in the loop the only calculations would be the forward and backward solve and then the calculation of the U^{n+1} from (1.1.9b). Since we were concentrating on the one space dimension case it occurred that there were not enough calculations inside the parallel part of the loop to notice a difference in timing between the serial and parallel versions.

The results, (see table 1.2.1), show that the PVM version, **par24-master.for** with **par24-worker.for**, gives comparable errors to the serial version, **par24-seq-doub.for**. Also, the results (see Table 1.2.2), obtained from comparing the serial to the parallel PVM version, show that the PVM version actually takes longer for problems involving smaller matrices, but as the matrices increase in size the PVM version runs about three times faster than the serial. The slower running at first can be attributed to two factors, the first is that PVM's use of the ethernet to send data back and forth causes some slow down. The second factor is that in the PVM there is more computing since each PVM statement is actually a function call.

Another consideration with PVM is that the current version does not allow quadruple precision. In order to see the algorithm approaching fourth order the program needs to be run in quadruple precision. An easy way around this is to run the serial version in quadruple precision to check order, and then run the serial version in double precision to compare with the PVM version. We cannot compare the PVM version with the quadruple precision version since the quadruple precision arithmetic takes longer than double precision arithmetic. It will become more apparent that we need quadruple precision to check the order when we try to obtain a sixth order algorithm to solve the wave equation.

Table 1.2.1: Results showing PVM version of algorithm, **par24-master.for** with **par24-worker.for**, gives the same error as serial version, **par24-seq-doub.for**.

h	k	Serial version ERROR	PVM version ERROR	$\widehat{p}$
$\frac{1}{10}$	$\frac{1}{10}$	4.47386E-05	4.47386E-05	
$\frac{1}{25}$	$\frac{1}{25}$	2.66376E-06	2.66376E-06	3.0788
$\frac{1}{50}$	$\frac{1}{50}$	2.41191E-07	2.41191E-07	3.4652
$\frac{1}{100}$	$\frac{1}{100}$	1.81852E-08	1.81849E-08	3.7294

Table 1.2.2: Results comparing the time it took the serial version, **par24-seq-doub.for**, and PVM version, **par24-master.for** with **par24-worker.for**, to complete the specified problem.

h	k	Time for Serial Version	Time for PVM Version
$\frac{1}{10}$	$\frac{1}{10}$	under 1 sec	3 secs
$\frac{1}{25}$	$\frac{1}{25}$	1 sec	8 secs
$\frac{1}{50}$	$\frac{1}{50}$	19 secs	21 secs
$\frac{1}{100}$	$\frac{1}{100}$	4 min 58 secs	2 min 12 secs

Chapter 2

The Wave Equation

2.1 A Fourth Order Scheme As A Special Case Of A General Second Order Scheme

We now look at the inhomogeneous wave equation

$$\begin{aligned} u_{tt} - \Delta u &= f(x,t) \quad \text{in } \Omega \times [0,\infty), \\ u &= 0 \quad \text{on } \partial\Omega \times [0,\infty), \\ u(x,0) = u_t(x,0) &= 0 \quad \text{in } \Omega. \end{aligned} \tag{2.1.1}$$

Once again we will find the function $u(x,t)$ that satisfies (2.1.1) for a given $f(x,t)$ where $\Omega = [0,1]$ and $t \in [0,1]$.

Dougalis, Makridakis, and Serbin will present the details of the development of the scheme we will use here. Since this has not yet appeared in the literature, we will sketch the main ideas leading to the algorithm we use, but we will leave the technicalities to them. For simplicity, we will stay with the wave equation, although, in general, $-\Delta$ may be replaced by any linear, second order, positive, elliptic operator L with time-independent coefficients.

The first algorithm we will work with is generally second order but is fourth order for certain values of the constants specified later. The algorithm is developed as follows. The standard Galerkin semidiscretization of (2.1.1) yields

$$\begin{aligned} u_{h,tt} + L_h u_h &= P_0 f \quad \text{in } \Omega \times [0,1], \\ u_h &= 0 \quad \text{on } \partial\Omega \times [0,1], \\ u_h(x,0) = u_{h,t}(x,0) &= 0 \quad \text{in } \Omega \end{aligned} \tag{2.1.2}$$

with $L_h = -\Delta_h$.

An application of Duhamel's principle in (2.1.2) gives the following

$$\begin{aligned} &u_h(t+2k) - 2\cos\left(kL_h^{1/2}\right) u_h(t+k) + u_h(t) \\ &\qquad = k\int_0^1 L_h^{-1/2} \sin\left(k(1-\tau)L_h^{1/2}\right) \left[P_0 f(t+k+k\tau) + P_0 f(t+k-k\tau)\right] d\tau. \end{aligned}$$

This is then approximated by

$$U^{n+2} - 2r\left(kL_h^{1/2}\right)U^{n+1} + U^n = kL_h^{-1/2}\sum_{i=1}^{m} q_i\left(kL_h^{1/2}\right)F_h^{n+1}(\tau_i k) \tag{2.1.3}$$

with $U^0 = 0$, U^1 given, where $U^n \approx u_h(t_n)$, and $F_h^{n+1}(\tau_i k) = P_0 f(t_{n+1} + \tau_i k) + P_0 f(t_{n+1} - \tau_i k)$, $t_n = nk$ with $r(\lambda)$ and $\{q_i(\lambda)\}_{i=1}^m$ appropriately selected rational functions and the τ_i's are to be chosen as distinct points in $[0,1]$. Clearly, $r\left(kL_h^{1/2}\right)$ is to approximate $\cos\left(kL_h^{1/2}\right)$, and the integral is approximated by a quadrature scheme.

We will say that this scheme is of order p (even) if and only if

(2.1.4a) $r(\lambda) = \cos\lambda + O\left(\lambda^{p+2}\right)$, $\lambda \to 0$ and for $0 \le 2l \le p+1$

(2.1.4b) $\sum_{i=1}^m \tau_i^{2l} q_i(\lambda) - \frac{(2l)!}{\lambda^{2l+1}}(-1)^{l+1}\left(\cos\lambda - \sum_{j=0}^{l}(-1)^j \frac{\lambda^{2j}}{(2j)!}\right) = O\left(\lambda^{p+1-2l}\right)$, $\lambda \to 0$, and $l = 0, 1, \ldots \frac{p}{2}$.

It turns out that in order for the scheme here to operate without very unrealistic compatibility assumptions on f, additional accuracy conditions will be imposed. Namely, with $m = \frac{p}{2}$, we insist that the left hand side of (2.1.4b) be identically zero for $l = 0, 1, \ldots \frac{p}{2} - 1$. This can be done, once $r(\lambda)$ and the $\{\tau_i\}_{i=1}^{p/2}$ are chosen, by solving a $\frac{p}{2} \times \frac{p}{2}$ system with Vandermonde matrix. One then shows that the left side of (2.1.4b) is also zero where $l = \frac{p}{2}$.

We first consider the case $m = 1$. An appropriate choice for $r(\lambda)$ is $\frac{1+(\beta-\frac{1}{2})\lambda^2}{1+\beta\lambda^2}$, with $r(\lambda) = \cos\lambda + O\left(\lambda^4\right)$ $(p = 2)$ for $\beta \neq \frac{1}{12}$ and $r(\lambda) = \cos\lambda + O\left(\lambda^6\right)$ $(p = 4)$ for $\beta = \frac{1}{12}$. Then, (2.1.3) is determined to be

$$(I - \beta k^2\Delta_h)[U^{n+2} + U^n] - 2[I - (\beta - \frac{1}{2})k^2\Delta_h]U^{n+1} = \frac{k^2}{2}[f_h(t_{n+1} + \theta k) + f_h(t_{n+1} - \theta k)] \tag{2.1.5}$$

where we have let $\tau_1 = \theta \in [0,1]$, and $f_h = P_0 f$. Rearranging, we obtain

$$(I - \beta k^2\Delta_h)[U^{n+2} + U^n - 2U^{n+1}] - k^2\Delta_h U^{n+1} = \frac{k^2}{2}[f_h(t_{n+1} + \theta k) + f_h(t_{n+1} - \theta k)]. \tag{2.1.6}$$

From (2.1.6) the algorithm is to first solve for Y^n in

$$(I - \beta k^2\Delta_h)[Y^n] = k^2\Delta_h U^{n+1} + \frac{k^2}{2}[f_h(t_{n+1} + \theta k) + f_h(t_{n+1} - \theta k)] \tag{2.1.7a}$$

then calculate

$$U^{n+2} = 2U^{n+1} - U^n + Y^n. \tag{2.1.7b}$$

In fact, if one selects $\theta = 0$ the computation simplifies. But, it will be of interest to see if one can obtain a method of order $p = 4$ without an appreciable extra cost by proper choices of θ and β. One verifies easily that with $\beta = \frac{1}{12}$, $r(\lambda) = \cos\lambda + O(\lambda^6)$.

According to (2.1.4b), (still using $m = 1$ and $r_1 = \theta$), we shall require that

$$\theta^{2l} q_1(\lambda) - \frac{(2l)!}{\lambda^{2l+1}}(-1)^{l+1}\left(\cos\lambda - \sum_{j=0}^{l}(-1)^j \frac{\lambda^{2j}}{(2j)!}\right) = O(\lambda^{5-2l})$$

as $\lambda \to 0$, and $l = 0, 1, 2$. With $l = 0$, thus, we require that

$$q_1(\lambda) = \frac{-1}{\lambda}[\cos\lambda - 1] + O(\lambda^5).$$

But since $\cos\lambda = r(\lambda) + O(\lambda^6)$, this says that we must have

$$q_1(\lambda) = \frac{-1}{\lambda}[r(\lambda) - 1 + O(\lambda^6)] + O(\lambda^5).$$

Hence, we choose

$$q_1(\lambda) = \frac{1 - r(\lambda)}{\lambda} = \frac{\frac{1}{2}\lambda}{(1 + \frac{1}{12}\lambda^2)}.$$

Now, for $l = 1$,

$$\begin{aligned}
\theta^2 q_1(\lambda) - \frac{2}{\lambda^3}\left\{\cos\lambda - \left[1 - \frac{\lambda^2}{2}\right]\right\} &= \theta^2\left\{\frac{\lambda}{2} - \frac{\lambda^3}{24} + O(\lambda^5)\right\} \\
&\quad - \frac{2}{\lambda^3}\left\{1 - \frac{\lambda^2}{2} + \frac{\lambda^4}{24} + O(\lambda^6) - 1 + \frac{\lambda^2}{2}\right\} \\
&= \theta^2\left\{\frac{\lambda}{2} - \frac{\lambda^3}{24} + O(\lambda^5)\right\} - \frac{2}{\lambda^3}\left\{\frac{\lambda^4}{24} + O(\lambda^6)\right\} \\
&= \theta^2\frac{\lambda}{2} - \frac{\lambda}{12} + O(\lambda^3)
\end{aligned}$$

In order for this expression to be $O(\lambda^3)$, we require $\theta^2\frac{\lambda}{2} - \frac{\lambda}{12} = 0$, i.e. $\frac{\theta^2}{2} = \frac{1}{12}$, or $\theta^2 = \frac{1}{6}$.

For the case $l = 2$,

$$\begin{aligned}
\theta^4 q_1(\lambda) + \frac{24}{\lambda^5}\left\{\cos\lambda - \left[1 - \frac{\lambda^2}{2} + \frac{\lambda^4}{24} + O(\lambda^6)\right]\right\} &= \theta^4 q_1(\lambda) + \frac{24}{\lambda^5}\{O(\lambda^6)\} \\
&= O(\lambda),
\end{aligned}$$

and the last condition is easily satisfied.

As before, we will test this scheme with $u(x,t) = t^\alpha x(1-x)$. This exact solution gives $F_i(t) = t^{\alpha-2}[\alpha(\alpha-1)x_i(1-x_i) + 2t^2]$. Our first thought was to use $\alpha = 3$, but, for the above u, this algorithm gives the exact solution for $\beta = \frac{1}{12}$ and $\theta = \frac{1}{\sqrt{6}}$. To see why we get the exact solution, we look at the general, say lth, coordinate of the left side minus the right side of (2.1.6). We abbreviate $x_l = \widehat{x}$ and use the exact solution to get $U_l^j = (x_l - x_l^2)t_j^\alpha$ and $(\triangle_h U^j)_l = u_{xx}(x_l, t_j)$. Then,

$$\begin{aligned}
&\left\{(I - \beta k^2 \triangle_h)[U^{n+2} + U^n - 2U^{n+1}] - k^2 \triangle_h U^{n+1} - \frac{k^2}{2}[f_h(t_{n+1} + \theta k) + f_h(t_{n+1} - \theta k)]\right\}_l \\
&= u(\widehat{x}, t_{n+2}) + u(\widehat{x}, t_n) - 2u(\widehat{x}, t_{n+1}) + \beta k^2 \left[u_{xx}(\widehat{x}, t_{n+2}) + u_{xx}(\widehat{x}, t_n) - 2u_{xx}(\widehat{x}, t_{n+1})\right] \\
&\quad + k^2 u_{xx}(\widehat{x}, t_{n+1}) - \frac{k^2}{2}\left[f(\widehat{x}, (n+1+\theta)k) + f(\widehat{x}, (n+1-\theta)k)\right] \\
&= [(n+2)k]^3 \widehat{x}(1-\widehat{x}) + [nk]^3 \widehat{x}(1-\widehat{x}) - 2[(n+1)k]^3 \widehat{x}(1-\widehat{x}) \\
&\quad + 2\beta k^2 \left[[(n+2)k]^3 + [nk]^3 - 2[(n+1)k]^3\right] + 2k^2[(n+1)k]^3 \\
&\quad - \frac{k^2}{2}\Big\{(n+1+\theta)k\left[6\widehat{x}(1-\widehat{x}) + 2[(n+1+\theta)k]^2\right] \\
&\qquad + (n+1-\theta)k\left[6\widehat{x}(1-\widehat{x}) + 2[(n+1-\theta)k]^2\right]\Big\} \\
&= (n^3 + 6n^2 + 12n + 8)k^3 \widehat{x}(1-\widehat{x}) + n^3 k^3 \widehat{x}(1-\widehat{x}) - 2(n^3 + 3n^2 + 3n + 1)k^3 \widehat{x}(1-\widehat{x}) \\
&\quad + 2\beta k^5 \left[n^3 + 6n^2 + 12n + 8 + n^3 - 2(n^3 + 3n^2 + 3n + 1)\right] \\
&\quad + 2k^5(n^3 + 3n^2 + 3n + 1) - \frac{k^2}{2}\Big\{(n+1+\theta)k6\widehat{x}(1-\widehat{x}) + 2[(n+1+\theta)k]^3 \\
&\qquad + (n+1-\theta)k6\widehat{x}(1-\widehat{x}) + 2[(n+1-\theta)k]^3\Big\} \\
&= 2k^5 n^3 + 6k^5 n^2 + (12\beta + 6)k^5 n + (12\beta + 2)k^5 - 3k^3 \theta \widehat{x}(1-\widehat{x}) + 3k^3 \theta \widehat{x}(1-\widehat{x}) \\
&\quad - 2n^3 k^5 - 6n^2 k^5 - 6nk^5 - 6n\theta^2 k^5 - 6\theta^2 k^5 - 2k^5 \\
&= \left(12\beta - 6\theta^2\right) nk^5 + \left(12\beta - 6\theta^2\right) k^5 \\
&= \left(12\beta - 6\theta^2\right)\left(nk^5 + k^5\right) \\
&= 0.
\end{aligned}$$

So, we get the exact solution when $12\beta - 6\theta^2 = 0$ or $\beta = \frac{\theta^2}{2}$. To avoid this (otherwise, we can claim nothing about the scheme here) we choose $\alpha = 4$.

Running the program, **hyp24-seq.for**, we get the results that verify that this method is generally second order. The results, also, show that this method is fourth order when $\theta = \frac{1}{\sqrt{6}}$ and $\beta = \frac{1}{12}$, (see Table 2.1.1). Table 2.1.2 shows we get similar results when the number of both the spatial and time subintervals are incremented.

2.2 A Sixth Order Scheme As A Special Case Of A General Fourth Order Scheme

Our next method involves a quadratic denominator for $r(\lambda)$. In general, let

$$r(\lambda) = \frac{1 + \left(\alpha_1 - \frac{1}{2}\right)\lambda^2 + \left(\alpha_2 - \frac{\alpha_1}{2} + \frac{1}{24}\right)\lambda^4}{1 + \alpha_1\lambda^2 + \alpha_2\lambda^4}.$$

Since

$$\begin{aligned}\cos\lambda\left\{1 + \alpha_1\lambda^2 + \alpha_2\lambda^4\right\} = 1 &+ \left(\alpha_1 - \frac{1}{2}\right)\lambda^2 + \left(\alpha_2 - \frac{\alpha_1}{2} + \frac{1}{24}\right)\lambda^4 \\ &+ \left(-\frac{1}{720} + \frac{\alpha_1}{24} - \frac{\alpha_2}{2}\right)\lambda^6 + \left(\frac{\alpha_2}{24} - \frac{\alpha_1}{720} + \frac{1}{40320}\right)\lambda^8 \\ &+ O\left(\lambda^{10}\right),\end{aligned}$$

it follows that

$$\cos\lambda - r(\lambda) = \frac{\left(-\frac{1}{720} + \frac{\alpha_1}{24} - \frac{\alpha_2}{2}\right)\lambda^6 + \left(\frac{\alpha_2}{24} - \frac{\alpha_1}{720} + \frac{1}{40320}\right)\lambda^8}{1 + \alpha_1\lambda^2 + \alpha_2\lambda^4} + O\left(\lambda^{10}\right). \qquad (2.2.1)$$

We have $\cos\lambda - r(\lambda) = O\left(\lambda^{\mu}\right)$, with $\mu = 6$ in general; for $\mu = 8$ we need $-\frac{1}{720} + \frac{\alpha_1}{24} - \frac{\alpha_2}{2} = 0$ or $\alpha_2 = \frac{30\alpha_1 - 1}{360}$, and $\mu = 10$ will arise if further we have $\alpha_1 = \frac{11}{252}$. In case we use $\alpha_1 = 2\beta$ and $\alpha_2 = \beta^2$, the denominator is a perfect square, and $\mu = 6$ in general, but $\mu = 8$ with $\beta = \frac{5 \pm \sqrt{15}}{60}$. So, we set $m = 2$, $p = 4$ in (2.1.4b). Namely we want

$$\sum_{i=1}^{2} \tau_i^{2l} q_i(\lambda) - \frac{(2l)!}{\lambda^{2l+1}}(-1)^{l+1}\left(\cos\lambda - \sum_{j=0}^{l}(-1)^j \frac{\lambda^{2j}}{(2j)!}\right) = O\left(\lambda^{5-2l}\right)$$

as $\lambda \to 0$, and $l = 0, 1, 2$.

Let us then define $q_1(\lambda)$ and $q_2(\lambda)$ by the 2×2 system

$$\sum_{i=1}^{2} \tau_i^{2l} q_i(\lambda) = \frac{(2l)!}{\lambda^{2l+1}}(-1)^{l+1}\left[r(\lambda) - \sum_{j=0}^{l}(-1)^j \frac{\lambda^{2j}}{(2j)!}\right].$$

Table 2.1.1: Results from program **hyp24-seq-quad.for** showing that this algorithm is generally second order but is fourth order for the specified values of β and θ

		$\beta = 0$	$\theta = 0$	$\beta = \frac{1}{12}$	$\theta = 0$	$\beta = \frac{1}{12}$	$\theta = 0$
h	k	ERROR	$\widehat{p}$	ERROR	$\widehat{p}$	ERROR	$\widehat{p}$
$\frac{1}{10}$	$\frac{1}{10}$	1.0500E-03		2.4991E-04		1.0666E-04	
$\frac{1}{10}$	$\frac{1}{20}$	2.6241E-04	2.0005	6.2489E-04	1.9997	2.5414E-05	2.0693
$\frac{1}{10}$	$\frac{1}{40}$	6.5609E-05	1.9999	1.5624E-04	1.9998	6.2756E-06	2.0178
$\frac{1}{10}$	$\frac{1}{80}$	1.6404E-05	1.9998	3.9061E-05	2.0000	1.5641E-06	2.0044
$\frac{1}{10}$	$\frac{1}{160}$	4.1011E-06	2.0000	9.7654E-06	2.0000	3.9072E-07	2.0011
$\frac{1}{10}$	$\frac{1}{320}$	1.0253E-06	2.0000	2.4414E-06	2.0000	9.7662E-08	2.0003

		$\beta = \frac{1}{12}$	$\theta = 0.5$	$\beta = \frac{1}{12}$	$\theta = 1$	$\beta = 0$	$\theta = \frac{1}{\sqrt{6}}$
h	k	ERROR	$\widehat{p}$	ERROR	$\widehat{p}$	ERROR	$\widehat{p}$
$\frac{1}{10}$	$\frac{1}{10}$	1.2402E-03		1.2495E-02		1.4514E-03	
$\frac{1}{10}$	$\frac{1}{20}$	3.1186E-04	1.9916	3.1244E-03	1.9997	3.6259E-04	2.0010
$\frac{1}{10}$	$\frac{1}{40}$	7.8081E-05	1.9979	7.8118E-04	1.9999	9.0633E-05	2.0002
$\frac{1}{10}$	$\frac{1}{80}$	1.9528E-05	1.9994	1.9530E-04	2.0000	2.2657E-05	2.0001
$\frac{1}{10}$	$\frac{1}{160}$	4.8826E-06	1.9998	4.8827E-05	1.9999	5.6643E-06	2.0000
$\frac{1}{10}$	$\frac{1}{320}$	1.2207E-06	1.9999	1.2207E-05	2.0000	1.4161E-06	2.0000

		$\beta = 0.1$	$\theta = \frac{1}{\sqrt{6}}$	$\beta = 0.5$	$\theta = \frac{1}{\sqrt{6}}$	$\beta = 1$	$\theta = \frac{1}{\sqrt{6}}$
h	k	ERROR	$\widehat{p}$	ERROR	$\widehat{p}$	ERROR	$\widehat{p}$
$\frac{1}{10}$	$\frac{1}{10}$	2.9659E-04		7.0412E-03		1.4934E-02	
$\frac{1}{10}$	$\frac{1}{20}$	7.2915E-05	2.0242	1.8000E-03	1.9678	3.9236E-03	1.9284
$\frac{1}{10}$	$\frac{1}{40}$	1.8152E-05	2.0061	4.5237E-04	1.9924	9.9306E-04	1.9822
$\frac{1}{10}$	$\frac{1}{80}$	4.5330E-06	2.0016	1.1324E-04	1.9981	2.4899E-04	1.9958
$\frac{1}{10}$	$\frac{1}{160}$	1.1330E-06	2.0003	2.8318E-05	1.9996	6.2292E-05	1.9990
$\frac{1}{10}$	$\frac{1}{320}$	2.8322E-07	2.0002	7.0802E-06	1.9999	1.5576E-05	1.9997

		$\beta = \frac{1}{12}$	$\theta = \frac{1}{\sqrt{6}}$
h	k	ERROR	$\widehat{p}$
$\frac{1}{10}$	$\frac{1}{10}$	6.9222E-06	
$\frac{1}{10}$	$\frac{1}{20}$	4.3290E-07	3.9991
$\frac{1}{10}$	$\frac{1}{40}$	2.7072E-08	3.9992
$\frac{1}{10}$	$\frac{1}{80}$	1.6924E-09	3.9997
$\frac{1}{10}$	$\frac{1}{160}$	1.0579E-10	3.9998
$\frac{1}{10}$	$\frac{1}{320}$	6.6125E-12	3.9999

Table 2.1.2: Results from program **hyp24-seq-quad.for** showing similar results as the number of space and time subintervals are both incremented.

		$\beta = \frac{1}{12}$	$\theta = \frac{1}{\sqrt{6}}$
h	k	ERROR	$\widehat{p}$
$\frac{1}{10}$	$\frac{1}{10}$	6.9222E-06	
$\frac{1}{20}$	$\frac{1}{20}$	4.3408E-07	3.9952
$\frac{1}{40}$	$\frac{1}{40}$	2.7129E-08	4.0001
$\frac{1}{80}$	$\frac{1}{80}$	1.6954E-09	4.0001
$\frac{1}{160}$	$\frac{1}{160}$	1.0596E-10	4.0000
$\frac{1}{320}$	$\frac{1}{320}$	6.6227E-12	4.0000

It follows, since $r(\lambda) = \cos\lambda + O(\lambda^6)$, that

$$\sum_{i=1}^{2} \tau_i^{2l} q_i(\lambda) = \frac{(2l)!}{\lambda^{2l+1}}(-1)^{l+1}\left[\cos\lambda + O(\lambda^6) - \sum_{j=0}^{l}(-1)^j \frac{\lambda^{2j}}{(2j)!}\right]$$

$$= \frac{(2l)!}{\lambda^{2l+1}}\left[\cos\lambda - \sum_{j=0}^{l}(-1)^j \frac{\lambda^{2j}}{(2j)!}\right] + O\left(\lambda^{6-(2l+1)}\right)$$

will then hold for $\lambda = 0, 1$.

To be specific, for $l = 0$ and $l = 1$, $q_1(\lambda)$ and $q_2(\lambda)$ are determined by

$$q_1(\lambda) + q_2(\lambda) = -\frac{1}{\lambda}[r(\lambda) - 1] \equiv S_1(\lambda)$$

$$\tau_1^2 q_1(\lambda) + \tau_2^2 q_2(\lambda) = \frac{2}{\lambda^3}\left[r(\lambda) - \left(1 - \frac{\lambda^2}{2}\right)\right] \equiv S_2(\lambda).$$

By direct computation

$$r(\lambda) - 1 = \frac{-\frac{1}{2}\lambda^2 + \left(\frac{1}{24} - \frac{\alpha_1}{2}\right)\lambda^4}{1 + \alpha_1\lambda^2 + \alpha_2\lambda^4}$$

and

$$r(\lambda) - 1 + \frac{\lambda^2}{2} = \frac{\frac{1}{24}\lambda^4 + \frac{\alpha_2}{2}\lambda^6}{1 + \alpha_1\lambda^2 + \alpha_2\lambda^4}.$$

So,

$$S_1(\lambda) \equiv -\frac{1}{\lambda}[r(\lambda) - 1] = \frac{\frac{1}{2}\lambda + \left(\frac{\alpha_1}{2} - \frac{1}{24}\right)\lambda^3}{1 + \alpha_1\lambda^2 + \alpha_2\lambda^4} = \frac{\lambda}{2} + O(\lambda^3) = O(\lambda)$$

and

$$S_2(\lambda) \equiv \frac{2}{\lambda^3}\left[r(\lambda) - \left(1 - \frac{\lambda^2}{2}\right)\right] = \frac{\frac{1}{12}\lambda + \alpha_2\lambda^3}{1 + \alpha_1\lambda^2 + \alpha_2\lambda^4} = \frac{\lambda}{12} + O(\lambda^3) = O(\lambda).$$

Solving the 2×2 system, we see that

$$q_1(\lambda) = \frac{\tau_2^2 S_1 - S_2}{\tau_2^2 - \tau_1^2}$$

$$q_2(\lambda) = \frac{S_2 - \tau_1^2 S_1}{\tau_2^2 - \tau_1^2}$$

and since $S_1 = O(\lambda)$ and $S_2 = O(\lambda)$ it follows that $q_1 = O(\lambda)$ and $q_2 = O(\lambda)$. Then for $l = 2$,

$$\tau_1^4 q_1(\lambda) + \tau_2^4 q_2(\lambda) + \frac{4!}{\lambda^5}\left[\cos\lambda - \left(1 - \frac{\lambda^2}{2} + \frac{\lambda^4}{4!}\right)\right] = O(\lambda)$$

as required.

The above method is fourth order. We have a choice of τ_1 and τ_2, and if we choose appropriately, we may obtain a sixth order method. First select $r(\lambda)$ so that $\cos\lambda = r(\lambda) + O\left(\lambda^8\right)$. We have suggested above ways to do this. Then require

$$\sum_{i=1}^{2} \tau_i^{2l} q_i(\lambda) = \frac{(2l)!}{\lambda^{2l+1}}(-1)^{l+1}\left[\cos\lambda - \sum_{j=0}^{l}(-1)^j \frac{\lambda^{2j}}{(2j)!}\right] + O\left(\lambda^{7-2l}\right), \qquad l = 0,1,2,3 \tag{2.2.2}$$

The cases $l = 0, 1$ follow from our previous choice of q_1 and q_2, and for $l = 2$,

$$\begin{aligned} \tau_1^4 q_1(\lambda) + \tau_2^4 q_2(\lambda) &= -\frac{4!}{\lambda^5}\left[1 - \frac{\lambda^2}{2!} + \frac{\lambda^4}{4!} - \frac{\lambda^6}{6!} + O\left(\lambda^8\right) - \left(1 - \frac{\lambda^2}{2} + \frac{\lambda^4}{4!}\right)\right] \\ &= \frac{\lambda}{30} + O\left(\lambda^3\right) \end{aligned} \tag{2.2.3}$$

So,

$$\begin{aligned} q_1(\lambda) &= \frac{1}{\tau_2^2 - \tau_1^2}\left[\tau_2^2 S_1 - S_2\right] \\ &= \frac{1}{\tau_2^2 - \tau_1^2}\left[\tau_2^2\left\{\frac{\lambda}{2} + O\left(\lambda^3\right)\right\} - \left\{\frac{\lambda}{12} + O\left(\lambda^3\right)\right\}\right] \\ &= \frac{1}{\tau_2^2 - \tau_1^2}\left[\left(\frac{\tau_2^2}{2} - \frac{1}{12}\right)\lambda + O\left(\lambda^3\right)\right], \\ q_2(\lambda) &= \frac{1}{\tau_2^2 - \tau_1^2}\left[S_2 - \tau_1^2 S_1\right] \\ &= \frac{1}{\tau_2^2 - \tau_1^2}\left[\frac{\lambda}{12} + O\left(\lambda^3\right) - \tau_1^2\left\{\frac{\lambda}{2} + O\left(\lambda^3\right)\right\}\right] \\ &= \frac{1}{\tau_2^2 - \tau_1^2}\left[\left(\frac{1}{12} - \frac{\tau_1^2}{2}\right)\lambda + O\left(\lambda^3\right)\right], \end{aligned}$$

and

$$\tau_1^4 q_1(\lambda) + \tau_2^4 q_2(\lambda) = \tau_1^4\left\{\frac{1}{\tau_2^2 - \tau_1^2}\left(\frac{\tau_2^2}{2} - \frac{1}{12}\right)\lambda\right\} + \tau_2^4\left\{\frac{1}{\tau_2^2 - \tau_1^2}\left(\frac{1}{12} - \frac{\tau_1^2}{2}\right)\lambda\right\} + O\left(\lambda^3\right).$$

We see that in order for (2.2.3) to hold, it must be that

$$\begin{aligned} &\frac{\tau_1^4}{\tau_2^2 - \tau_1^2}\left(\frac{\tau_2^2}{2} - \frac{1}{12}\right) + \frac{\tau_2^4}{\tau_2^2 - \tau_1^2}\left(\frac{1}{12} - \frac{\tau_1^2}{2}\right) = \frac{1}{30} \\ \iff &\frac{1}{12\left(\tau_2^2 - \tau_1^2\right)}\left[\tau_1^4\left(6\tau_2^2 - 1\right) + \tau_2^4\left(1 - 6\tau_1^2\right)\right] = \frac{1}{30} \\ \iff &\frac{1}{12\left(\tau_2^2 - \tau_1^2\right)}\left[6\tau_1^4\tau_2^2 - \tau_1^4 + \tau_2^4 - 6\tau_1^2\tau_2^4\right] = \frac{1}{30} \\ \iff &\frac{1}{12\left(\tau_2^2 - \tau_1^2\right)}\left[6\tau_1^2\tau_2^2\left(\tau_1^2 - \tau_2^2\right) - \left(\tau_1^4 - \tau_2^4\right)\right] = \frac{1}{30} \\ \iff &\frac{\tau_1^2 - \tau_2^2}{12\left(\tau_2^2 - \tau_1^2\right)}\left[6\tau_1^2\tau_2^2 - \left(\tau_1^2 + \tau_2^2\right)\right] = \frac{1}{30} \end{aligned}$$

$$\iff -\frac{1}{12}\left[6\tau_1^2\tau_2^2 - (\tau_1^2+\tau_2^2)\right] = \frac{1}{30}$$

$$\iff 6\tau_1^2\tau_2^2 - (\tau_1^2+\tau_2^2) = -\frac{2}{5}.$$

Now, for convenience, let $\tau_2 = \theta$. Then, we require

$$6\tau_1^2\theta^2 - (\tau_1^2+\theta^2) = -\frac{2}{5}$$

or

$$\tau_1^2\left[6\theta^2 - 1\right] = \theta^2 - \frac{2}{5}$$

so

$$\tau_1^2 = \frac{5\theta^2-2}{5\left[6\theta^2-1\right]} = \frac{2-5\theta^2}{5\left[1-6\theta^2\right]}.$$

Simplifying, letting $\eta = \theta^2$, the condition becomes

$$\tau_1^2 = \frac{2-5\eta}{5\left[1-6\eta\right]}.$$

Surely, τ_1^2 must be greater than or equal to 0, and we ask further that $0 \leq \theta \leq 1$ to fulfill our request that $0 \leq \tau_i \leq 1$. It is easy to show that either $\eta \geq \frac{2}{5}$ or $\eta \leq \frac{3}{25}$ (along with $\eta \geq 0$) should be required.

The case where $l = 3$ falls out easily:

$$\begin{aligned}\tau_1^6 q_1(\lambda) + \tau_2^6 q_2(\lambda) - \frac{6!}{\lambda^7}\left[\cos\lambda - \left(1 - \frac{\lambda^2}{2!} + \frac{\lambda^4}{4!} - \frac{\lambda^6}{6!}\right)\right] \\ = \tau_1^6 q_1(\lambda) + \tau_2^6 q_2(\lambda) - \frac{6!}{\lambda^7}\left[O\left(\lambda^8\right)\right] \\ = O(\lambda)\end{aligned}$$

as required.

We now discuss the implementation of these schemes. With $r(\lambda)$ as defined above, we obtain

$$\begin{aligned} r\left(kL_h^{1/2}\right) &= \left(I + \alpha_1 k^2 L_h + \alpha_2 k^4 L_h^2\right)^{-1}\left[I + \left(\alpha_1 - \frac{1}{2}\right)k^2 L_h + \left(\alpha_2 - \frac{\alpha_1}{2} + \frac{1}{24}\right)k^4 L_h^2\right] \\ &= I + \left(I + \alpha_1 k^2 L_h + \alpha_2 k^4 L_h^2\right)^{-1}\left[\left(\frac{-1}{2}\right)k^2 L_h + \left(\frac{1}{24} - \frac{\alpha_1}{2}\right)k^4 L_h^2\right] \end{aligned} \tag{2.2.4}$$

and

$$kL_h^{-1/2}q_1\left(kL_h^{1/2}\right) = \frac{1}{\tau_2^2-\tau_1^2}\left(I+\alpha_1k^2L_h+\alpha_2k^4L_h^2\right)^{-1}\left\{\tau_2^2\left[\frac{1}{2}k^2I+\left(\frac{\alpha_1}{2}-\frac{1}{24}\right)k^4L_h\right]\right.$$
$$\left.-\left[\frac{1}{12}k^2I+\alpha_2k^4L_h\right]\right\} \tag{2.2.5a}$$

$$kL_h^{-1/2}q_2\left(kL_h^{1/2}\right) = \frac{1}{\tau_2^2-\tau_1^2}\left(I+\alpha_1k^2L_h+\alpha_2k^4L_h^2\right)^{-1}\left\{\left[\frac{1}{12}k^2I+\alpha_2k^4L_h\right]\right\}$$
$$-\tau_1^2\left[\frac{1}{2}k^2I+\left(\frac{\alpha_1}{2}-\frac{1}{24}\right)k^4L_h\right]. \tag{2.2.5b}$$

Using (2.2.4) and (2.2.5), (2.1.3) becomes

$$\begin{aligned}U^{n+2}+U^n = 2\Big[I+\left(I+\alpha_1k^2L_h+\alpha_2k^4L_h^2\right)^{-1}\Big\{&\left(\frac{-1}{2}\right)k^2L_h\\ &+\left(\frac{1}{24}-\frac{\alpha_1}{2}\right)k^4L_h^2\Big\}\Big]U^{n+1}\\ +\frac{1}{\tau_2^2-\tau_1^2}\left(I+\alpha_1k^2L_h+\alpha_2k^4L_h^2\right)^{-1}&\left(\left\{\tau_2^2\left[\frac{1}{2}k^2I+\left(\frac{\alpha_1}{2}-\frac{1}{24}\right)k^4L_h\right]\right.\right.\\ &\left.-\left[\frac{1}{12}k^2I+\alpha_2k^4L_h\right]\right\}F_h^{n+1}(\tau_1k)\\ +\Big\{\Big[\frac{1}{12}k^2I+\alpha_2k^4L_h\Big]&\\ &\left.\left.-\tau_1^2\left[\frac{1}{2}k^2I+\left(\frac{\alpha_1}{2}-\frac{1}{24}\right)k^4L_h\right]\right\}F_h^{n+1}(\tau_2k)\right)\end{aligned}$$

which is

$$\begin{aligned}U^{n+2}+U^n-2U^{n+1}+\left(I+\alpha_1k^2L_h+\alpha_2k^4L_h^2\right)^{-1}&\left\{k^2L_h+\left(\frac{-1}{12}+\alpha_1\right)k^4L_h^2\right\}U^{n+1}\\ =\frac{1}{\tau_2^2-\tau_1^2}\left(I+\alpha_1k^2L_h+\alpha_2k^4L_h^2\right)^{-1}&\left(\left\{\tau_2^2\left[\frac{1}{2}k^2I+\left(\frac{\alpha_1}{2}-\frac{1}{24}\right)k^4L_h\right]\right.\right.\\ &\left.-\left[\frac{1}{12}k^2I+\alpha_2k^4L_h\right]\right\}F_h^{n+1}(\tau_1k)\\ +\Big\{\Big[\frac{1}{12}k^2I+\alpha_2k^4L_h\Big]&\\ &\left.\left.-\tau_1^2\left[\frac{1}{2}k^2I+\left(\frac{\alpha_1}{2}-\frac{1}{24}\right)k^4L_h\right]\right\}F_h^{n+1}(\tau_2k)\right)\end{aligned}$$

or

$$\begin{aligned}\left(I+\alpha_1k^2L_h+\alpha_2k^4L_h^2\right)&\left[U^{n+2}-2U^{n+1}+U^n\right]+\left[k^2L_h+\left(\frac{-1}{12}+\alpha_1\right)k^4L_h^2\right]U^{n+1}\\ &=\frac{1}{\tau_2^2-\tau_1^2}\left(\left\{\tau_2^2\left[\frac{1}{2}k^2I+\left(\frac{\alpha_1}{2}-\frac{1}{24}\right)k^4L_h\right]\right.\right.\\ &\quad\left.-\left[\frac{1}{12}k^2I+\alpha_2k^4L_h\right]\right\}F_h^{n+1}(\tau_1k)\\ &\quad+\Big\{\Big[\frac{1}{12}k^2I+\alpha_2k^4L_h\Big]\\ &\quad\left.\left.-\tau_1^2\left[\frac{1}{2}k^2I+\left(\frac{\alpha_1}{2}-\frac{1}{24}\right)k^4L_h\right]\right\}F_h^{n+1}(\tau_2k)\right).\end{aligned} \tag{2.2.6}$$

From (2.2.6), with $L_h = -\Delta_h$, we proceed by solving for Y^n in

$$\begin{aligned}(I-\beta_1 k^2\Delta_h)\,(I-\beta_2 k^2\Delta_h)\,Y^n = &\left(-k^2\Delta_h + \left(\alpha_1 - \frac{1}{12}\right)k^4\Delta_h^2\right)U^{n+1} \\ &+ \frac{1}{\tau_2^2-\tau_1^2}\left[\tau_2^2\left\{\frac{1}{2}k^2 - \left(\frac{\alpha_1}{2}-\frac{1}{24}\right)k^4\Delta_h\right\}\right. \\ &\left.-\left\{\frac{1}{12}k^2 I - \alpha_2 k^4\Delta_h\right\}\right]F_h^{n+1}(\tau_1 k) \qquad (2.2.7a) \\ &+ \frac{1}{\tau_2^2-\tau_1^2}\left[\left\{\frac{1}{12}k^2 I - \alpha_2 k^4\Delta_h\right\}\right. \\ &\left.-\tau_1^2\left\{\frac{1}{2}k^2 - \left(\frac{\alpha_1}{2}-\frac{1}{24}\right)k^4\Delta_h\right\}\right]F_h^{n+1}(\tau_2 k)\end{aligned}$$

where

$$(I-\alpha_1 k^2\Delta_h + \alpha_2 k^4\Delta_h^2) = (I-\beta_1 k^2\Delta_h)\,(I-\beta_2 k^2\Delta_h)\,,$$

with $\alpha_1 = \beta_1 + \beta_2$ and $\alpha_2 = \beta_1\beta_2$, and

$$Y^n = U^{n+2} - 2U^{n+1} + U^n.$$

Also,

$$F_h^{n+1}(\tau_j k) = P_0 f(t_{n+1} + \tau_j k) + P_0 f(t_{n+1} - \tau_j k)\,.$$

We then find

$$U^{n+2} = Y^n + 2U^{n+1} - U^n. \qquad (2.2.7b)$$

For this to be sixth order we need to choose β_1 and β_2 so that $-\frac{1}{720} + \frac{\beta_1+\beta_2}{24} - \frac{\beta_1\beta_2}{2} = 0$ or $\beta_2 = \frac{30\beta_1-1}{360\beta_1-30}$. Recall, also, we need to choose $\tau_1^2 = \frac{2-5\tau_2^2}{5(1-6\tau_2^2)}$ with $\tau_2^2 < \frac{3}{25}$ and $\tau_2^2 > \frac{2}{5}$.

The results, (see Table 2.2.1), obtained when we run the program **hyp46-seq-quad.for** clearly show what we expect. A sixth order scheme is obtained when τ_1 and τ_2 are determined as above, and β_1 and β_2 are either both equal to $\frac{5\pm\sqrt{15}}{60}$ or are determined as above. The results clearly illustrate that if both the τ's and the β's do not satisfy these conditions, then we obtain a fourth order scheme. Table 2.2.2 shows we get similar results when the number of both the spatial and time subintervals are incremented, so order reduction is not apparent

Table 2.2.1: Results from program **hyp46-seq-quad.for** showing that this algorithm is generally fourth order, but it is sixth order for the specified values of β_i and τ_i. (note: when $\beta_1 = \frac{3}{20}$ then $\beta_2 = .14583\overline{3}$)

		$\beta_1 = 0.5$	$\beta_2 = 0.5$	$\beta_1 = \frac{5+\sqrt{15}}{60}$ $\beta_2 = \frac{5+\sqrt{15}}{60}$		$\beta_1 = \frac{5+\sqrt{15}}{60}$ $\beta_2 = \frac{5+\sqrt{15}}{60}$	
		$\tau_1 = 0.5$	$\tau_2 = 0.1$	$\tau_1 = \sqrt{\frac{2-5\tau_2^2}{5[1-6\tau_2^2]}}$ $\tau_2 = 0.1$		$\tau_1 = 0.5$	$\tau_2 = 0.1$
h	k	ERROR	$\widehat{p}$	ERROR	$\widehat{p}$	ERROR	$\widehat{p}$
$\frac{1}{10}$	$\frac{1}{10}$	1.8620E-04		1.1289E-07		1.3861E-06	
$\frac{1}{10}$	$\frac{1}{20}$	1.2346E-05	3.9148	1.7542E-09	6.0080	8.2098E-08	4.0776
$\frac{1}{10}$	$\frac{1}{40}$	7.8377E-07	3.9774	2.7106E-11	6.0160	5.0605E-09	4.0200
$\frac{1}{10}$	$\frac{1}{80}$	4.9180E-08	3.9943	4.2475E-13	5.9959	3.1519E-10	4.0050
$\frac{1}{10}$	$\frac{1}{160}$	3.0770E-09	3.9985	6.6639E-15	5.9941	1.9684E-11	4.0012
$\frac{1}{10}$	$\frac{1}{320}$	1.9237E-10	3.9996	1.0432E-16	5.9973	1.2300E-12	4.0002

		$\beta_1 = \frac{3}{20}$, $\beta_2 = \frac{30\beta_1-1}{360\beta_1-30}$		$\beta_1 = \frac{3}{20}$, $\beta_2 = \frac{30\beta_1-1}{360\beta_1-30}$		$\beta_1 = 0.151$, $\beta_2 = \frac{30\beta_1-1}{360\beta_1-30}$	
		$\tau_1 = \sqrt{\frac{2-5\tau_2^2}{5[1-6\tau_2^2]}}$, $\tau_2 = 0.9$		$\tau_1 = \sqrt{\frac{2-5\tau_2^2}{5[1-6\tau_2^2]}}$, $\tau_2 = 0.1$		$\tau_1 = \sqrt{\frac{2-5\tau_2^2}{5[1-6\tau_2^2]}}$, $\tau_2 = 0.1$	
h	k	ERROR	$\widehat{p}$	ERROR	$\widehat{p}$	ERROR	$\widehat{p}$
$\frac{1}{10}$	$\frac{1}{10}$	9.5166E-08		1.1292E-07		1.1295E-07	
$\frac{1}{10}$	$\frac{1}{20}$	1.4788E-09	6.0080	1.7546E-09	6.0080	1.7552E-09	6.0080
$\frac{1}{10}$	$\frac{1}{40}$	2.2850E-11	6.0160	2.7113E-11	6.0160	2.7121E-11	6.0160
$\frac{1}{10}$	$\frac{1}{80}$	3.5806E-13	5.9959	4.2486E-13	5.9959	4.2498E-13	5.9959
$\frac{1}{10}$	$\frac{1}{160}$	5.6177E-15	5.9941	6.6656E-15	5.9941	6.6676E-15	5.9941
$\frac{1}{10}$	$\frac{1}{320}$	8.7941E-17	5.9973	1.0435E-16	5.9973	1.0438E-16	5.9973

		$\beta_1 = 0.16$, $\beta_2 = \frac{30\beta_1-1}{360\beta_1-30}$		$\beta_1 = 0.25$, $\beta_2 = \frac{30\beta_1-1}{360\beta_1-30}$		$\beta_1 = 0.5$, $\beta_2 = \frac{30\beta_1-1}{360\beta_1-30}$	
		$\tau_1 = \sqrt{\frac{2-5\tau_2^2}{5[1-6\tau_2^2]}}$, $\tau_2 = 0.1$		$\tau_1 = \sqrt{\frac{2-5\tau_2^2}{5[1-6\tau_2^2]}}$, $\tau_2 = 0.1$		$\tau_1 = \sqrt{\frac{2-5\tau_2^2}{5[1-6\tau_2^2]}}$, $\tau_2 = 0.1$	
h	k	ERROR	$\widehat{p}$	ERROR	$\widehat{p}$	ERROR	$\widehat{p}$
$\frac{1}{10}$	$\frac{1}{10}$	1.1369E-07		1.3874E-07		2.3228E-07	
$\frac{1}{10}$	$\frac{1}{20}$	1.7670E-09	6.0076	2.1722E-09	5.9971	3.7294E-09	5.9608
$\frac{1}{10}$	$\frac{1}{40}$	2.7305E-11	6.0160	3.3611E-11	6.0141	5.8008E-11	6.0066
$\frac{1}{10}$	$\frac{1}{80}$	4.2787E-13	5.9959	5.2669E-13	5.9958	9.0914E-13	5.9956
$\frac{1}{10}$	$\frac{1}{160}$	6.7129E-15	5.9941	8.2648E-15	5.9938	1.4276E-14	5.9928
$\frac{1}{10}$	$\frac{1}{320}$	1.0509E-16	5.9973	1.2939E-16	5.9972	2.2353E-16	5.9970

		$\beta_1 = \frac{3}{20}$, $\beta_2 = \frac{30\beta_1-1}{360\beta_1-30}$		$\beta_1 = \frac{3}{20}$, $\beta_2 = 0.145$		$\beta_1 = \frac{3}{20}$, $\beta_2 = 0.146$	
		$\tau_1 = 0.5$, $\tau_2 = 0.1$		$\tau_1 = \sqrt{\frac{2-5\tau_2^2}{5[1-6\tau_2^2]}}$, $\tau_2 = 0.1$		$\tau_1 = \sqrt{\frac{2-5\tau_2^2}{5[1-6\tau_2^2]}}$, $\tau_2 = 0.1$	
h	k	ERROR	$\widehat{p}$	ERROR	$\widehat{p}$	ERROR	$\widehat{p}$
$\frac{1}{10}$	$\frac{1}{10}$	1.3861E-06		1.6953E-07		1.0204E-07	
$\frac{1}{10}$	$\frac{1}{20}$	8.2098E-08	4.0776	5.6340E-09	4.9112	1.1570E-09	6.4626
$\frac{1}{10}$	$\frac{1}{40}$	5.0605E-09	4.0200	2.8344E-10	4.3131	2.7954E-11	5.3712
$\frac{1}{10}$	$\frac{1}{80}$	3.1519E-10	4.0050	1.6620E-11	4.0921	2.8699E-12	3.2840
$\frac{1}{10}$	$\frac{1}{160}$	1.9684E-11	4.0012	1.0215E-12	4.0242	1.9715E-13	3.8636
$\frac{1}{10}$	$\frac{1}{320}$	1.2300E-12	4.0002	6.3571E-14	4.0061	1.2602E-14	3.9676

Table 2.2.2: Results from program **hyp46-seq-quad.for** showing similar results as the number of time and space subintervals are both incremented.

$\beta_1 = \frac{3}{20}$, $\beta_2 = \frac{30\beta_1 - 1}{360\beta_1 - 30}$

$\tau_1 = \sqrt{\frac{2 - 5\tau_2^2}{5[1 - 6\tau_2^2]}}$, $\tau_2 = 0.99$

h	k	ERROR	$\widehat{p}$
$\frac{1}{10}$	$\frac{1}{10}$	8.8563E-08	
$\frac{1}{25}$	$\frac{1}{25}$	3.7181E-10	5.9731
$\frac{1}{50}$	$\frac{1}{50}$	5.8424E-12	5.9919
$\frac{1}{100}$	$\frac{1}{100}$	9.2000E-14	5.9888

2.3 Obtaining A Sixth Order Parallel Algorithm

To come up with the partial fraction approach we substitute

$$r\left(kL_h^{1/2}\right) = A_0 + A_1\left(I+\beta_1 k^2 L_h\right)^{-1} + A_2\left(I+\beta_2 k^2 L_h\right)^{-1} \tag{2.3.1a}$$

and

$$kL_h^{-1/2} q_i\left(kL_h^{1/2}\right) = \delta_1^{(i)}\left(I+\beta_1 k^2 L_h\right)^{-1} + \delta_2^{(i)}\left(I+\beta_2 k^2 L_h\right)^{-1} \tag{2.3.1b}$$

into (2.1.3) and get

$$\begin{aligned} U^{n+2} + U^n &= 2\left[A_0 + A_1\left(I+\beta_1 k^2 L_h\right)^{-1} + A_2\left(I+\beta_2 k^2 L_h\right)^{-1}\right] U^{n+1} \\ &\quad + k^2 \sum_{i=1}^{2}\left[\delta_1^{(i)}\left(I+\beta_1 k^2 L_h\right)^{-1} + \delta_2^{(i)}\left(I+\beta_2 k^2 L_h\right)^{-1}\right] F_h^{n+1}(\tau_i k) \\ &= 2A_0 U^{n+1} + 2A_1\left(I+\beta_1 k^2 L_h\right)^{-1} U^{n+1} + 2A_2\left(I+\beta_2 k^2 L_h\right)^{-1} U^{n+1} \\ &\quad + k^2\left[\delta_1^{(1)}\left(I+\beta_1 k^2 L_h\right)^{-1} F_h^{n+1}(\tau_1 k) + \delta_2^{(1)}\left(I+\beta_2 k^2 L_h\right)^{-1} F_h^{n+1}(\tau_1 k)\right. \\ &\quad \left. + k^2\left[\delta_1^{(2)}\left(I+\beta_1 k^2 L_h\right)^{-1} F_h^{n+1}(\tau_2 k) + \delta_2^{(2)}\left(I+\beta_2 k^2 L_h\right)^{-1} F_h^{n+1}(\tau_2 k)\right]\right]. \end{aligned}$$

Thus,

$$U^{n+2} - 2A_0 U^{n+1} + U^n = \left(I+\beta_1 k^2 L_h\right)^{-1}\Phi_1 + \left(I+\beta_2 k^2 L_h\right)^{-1}\Phi_2 \tag{2.3.2a}$$

where

$$\Phi_i = 2A_i U^{n+1} + k^2\delta_i^{(1)} F_h^{n+1}(\tau_1 k) + k^2\delta_i^{(2)} F_h^{n+1}(\tau_2 k)\,. \tag{2.3.2b}$$

In actually implementing (2.3.2) we let $D_i = \delta_i^{(1)}$ and $C_i = \delta_i^{(2)}$. The A's, C's, and D's are found using partial fractions via (2.3.1) from elementary calculus. Hence, we get the following constants:

$$\begin{aligned} A_0 &= \frac{\alpha_2 - \frac{\alpha_1}{2} + \frac{1}{24}}{\alpha_2} \\ A_1 &= \frac{1-12\beta_2}{24\beta_1(\beta_1-\beta_2)} \qquad A_2 = \frac{1-12\beta_1}{24\beta_2(\beta_2-\beta_1)} \\ D_1 = \delta_1^{(1)} &= \frac{k^2}{(\tau_2^2-\tau_1^2)(\beta_1-\beta_2)}\left[\left(\frac{\alpha_1}{2} - \frac{1}{24} - \frac{\beta_2}{2}\right)\tau_2^2 - \alpha_2 + \frac{\beta_2}{12}\right] \\ D_2 = \delta_2^{(1)} &= \frac{k^2}{(\tau_2^2-\tau_1^2)(\beta_1-\beta_2)}\left[\left(\frac{\beta_1}{2} - \frac{\alpha_1}{2} + \frac{1}{24}\right)\tau_2^2 + \alpha_2 - \frac{\beta_1}{12}\right] \\ C_1 = \delta_1^{(2)} &= \frac{k^2}{(\tau_2^2-\tau_1^2)(\beta_1-\beta_2)}\left[\alpha_2 - \frac{\beta_2}{12} - \tau_1^2\left(\frac{\alpha_1}{2} - \frac{1}{24} - \frac{\beta_2}{2}\right)\right] \\ C_2 = \delta_2^{(2)} &= \frac{k^2}{(\tau_2^2-\tau_1^2)(\beta_1-\beta_2)}\left[\frac{\beta_1}{12} - \alpha_2 + \tau_1^2\left(\frac{\alpha_1}{2} - \frac{1}{24} - \frac{\beta_1}{2}\right)\right]. \end{aligned}$$

The algorithm itself is actually performed by first solving

$$\left(I + \beta_i k^2 L_h\right) Y^i = \Phi_i, \qquad where \quad i = 1, 2, \tag{2.3.3a}$$

for Y^i; and then, we let

$$W^n = Y^1 + Y^2 \tag{2.3.3b}$$

and

$$U^{n+2} = W^n + 2A_0 U^{n+1} - U^n. \tag{2.3.3c}$$

This algorithm is suitable for parallelism, since one equation in (2.3.3a) can be solved independently of the other equation. The results can then be gathered to calculate (2.3.3b) and (2.3.3c).

The first results, (see Table 2.3.1), we look at are comparisons between the original serial version based on algorithm (2.2.7), program **hyp46-seq-doub.for**; the original partial fraction version based on algorithm (2.3.3), program **hyp46-pf-doub.for**; and the PVM version based on (2.3.3), program **hyp-master.for** with **hyp-worker.for**. These results show that we get comparable errors for the specified h and k. Also, we see that order reduction is not apparent in the PVM version. The second set of results, (see Table 2.3.2), show, as with the parabolic problem, that as we decrease h or increase the matrix size, the PVM version will eventually determine a solution faster than the serial version.

Table 2.3.1: Results showing that the partial fractions version in serial mode, **hyp46-pf-doub.for**, and the corresponding parallel version, **hyp-master.for** with **hyp-worker.for**, give comparable error values to the original serial version, **hyp46-seq-doub.for**.

$\beta_1 = 0.15$, $\beta_2 = \frac{30\beta_1 - 1}{360\beta_1 - 30}$, $\tau_1 = \sqrt{\frac{2-5\tau_2^2}{5[1-6\tau_2^2]}}$, $\tau_2 = 0.99$

h	k	Original Serial Version ERROR	Partial Fractions Version in Serial ERROR	PVM Version ERROR	$\widehat{p}$
$\frac{1}{10}$	$\frac{1}{10}$	8.8563E-08	8.8563E-08	8.8563E-08	
$\frac{1}{25}$	$\frac{1}{25}$	3.7181E-10	3.7180E-10	3.7180E-10	5.9731
$\frac{1}{50}$	$\frac{1}{50}$	5.8424E-12	5.7784E-12	5.7784E-12	6.0077
$\frac{1}{100}$	$\frac{1}{100}$	9.20E-14	8.588E-13	8.588E-13	

Table 2.3.2: Results showing the time comparisons for finishing certain problems by the serial version, **hyp46-seq-doub.for**; partial fraction version in serial mode, **hyp46-pf-doub.for**; and the corresponding PVM version, **hyp-master.for** with **hyp-worker.for**.

h	k	Time for Serial Version	Time for Partial Fraction Version in Serial	Time for PVM Version
$\frac{1}{10}$	$\frac{1}{10}$	under 1 sec	under 1 sec	3-6 secs
$\frac{1}{25}$	$\frac{1}{25}$	1 sec	1 sec	8 secs
$\frac{1}{50}$	$\frac{1}{50}$	12 sec	12 sec	20 secs
$\frac{1}{100}$	$\frac{1}{100}$	3 min 30 sec	3 min 6 sec	2 min 17 sec

Chapter 3

Conclusion

In this thesis, we have successfully taken schemes for the heat and wave equations and developed parallel code in PVM. We have been able to make sure these codes gave the error and order we expected, so order reduction was not evidenced.

Some further directions that could be taken with these schemes is to do more work with the codes mentioned in section 1.2 which make use of the LL^t decomposition outside the loop. Also, we could pursue two and higher space dimensions to see what effect this would have on the time of the two parallel versions compared to their corresponding serial versions. Other types of problems could also be looked at to determine if partial fractions could be used to come up with an algorithm suitable for parallelism. Finally, the parallel code could be ported over to an actual parallel machine to check the performance.

LIST OF REFERENCES

LIST OF REFERENCES

[1] Brenner, P., Crouzeix, M., and Thomée, V., *Single Step Methods for Inhomogeneous Linear Differential Equations in Banuch Space*, RAIRO, Analyse Numérique, 16 (1982), pp. 5-26.

[2] Burden, Richard L. and Faires, J. Douglas, *Numerical Analysis*, 4th edition, PWS-Kent, 1989.

[3] Dougalis and Makridakis, private conversations.

[4] Golub, Gene H. and Van Loan, Charles F., *Matrix Computations*, The Johns Hopkins University Press, 1989.

[5] Norsett, S. P. and Wanner, G., *The Real-Pole Sandwich for Rational Approximations and Oscillation Equations*, BIT, 19 (1979), pp. 79-94.

[6] Oak Ridge National Laboratory, *PVM 3 User's Guide and Reference Manual*, (1994).

[7] Serbin, Steven M., *A Scheme for Parallelizing Certain Algorithms for the Linear Inhomogeneous Heat Equation*, Siam J. Sci. Stat. Comput., vol. 13, no. 2 (1992), pp. 449-458.

[8] Verwer, J. G., *Convergence and Order Reduction of Diagonally Implicit Runge-Kutta Schemes in the Method of Lines*, Pitnam Research Notes in Mathematics, Numerical Analysis, 140 (1986), pp. 220-237.

APPENDIX

Appendix A

PVM Basics

A-1. Running PVM

This tutorial should help get someone up and running with PVM. It talks about some of the more useful commands when running PVM and some of the FORTRAN functions calls used in programming with PVM. It will show you how to get your virtual machine set up and FORTRAN programs running with PVM. If there are any more questions you should check out the **PVM 3 USER'S GUIDE AND REFERENCE MANUAL**, [6], which some of this is taken from.

1. To be able to access other machines on the mathsun network, you need to create a file **.rhosts** with the name of the machine you are logged in on followed by your user name. If I were logged in on mathsun15, my **.rhosts** file would be

   mathsun15 fister

2. To run PVM you need to be in your home directory. Type **pvm** to start PVM.

3. At the PVM prompt type **conf**. This will list the machine that you are already logged in on. After setting up your virtual machine you can type **conf** and the other machines you have added will also be listed.

4. To add a machine you would use the **add** command at the PVM prompt. The **delete** command would be used to delete a machine from your virtual machine.

   pvm> add mathsun8 will set add mathsun8 to your virtual machine.

5. To leave PVM running, so you can run parallel programs, type **quit** at the PVM prompt. This will keep PVM running as a background project.

6. When compiling FORTRAN programs you need to link them to the files **pvm3/lib/SUN4/libfpvm3.a** and **pvm3/lib/SUN4/libpvm3.a**. To run programs on the SUN both the master and worker executables should be located in the directory **pvm3/bin/SUN4**. The easiest way I find to do this is to create a script file **mkmaster** which compiles and links the appropriate programs and then moves the executables to the directory **pvm3/bin/SUN4**. All you need to type is **csh mkmaster**. An example of one of these files is

   ```
   cp master.for thesis/
   cp worker.for thesis/
   f77 -o master master.for /pvm3/lib/SUN4/libfpvm3.a /pvm3/lib/SUN4/libpvm3.a
   mv master /pvm3/bin/SUN4/master
   f77 -o worker worker.for /pvm3/lib/SUN4/libfpvm3.a /pvm3/lib/SUN4/libpvm3.a
   mv worker /pvm3/bin/SUN4/worker
   ```

7. Now you can change to the directory **/pvm3/bin/SUN4/** and run the program by typing the name of the master program.

8. When you are through running the programs you need to get back to your home directory and type **pvm** to get back in PVM. Then type **halt**. This will kill PVM for this session.

A-2. The PVM Commands Used In This Thesis

These are the PVM commands used in this thesis. Most are self-explanatory, especially, if you look at their usage in the included programs. For my information on their usage see the **PVM 3 USER'S GUIDE AND REFERENCE MANUAL**, [6].

pvmfmytid : Enrolls in PVM.

pvmfspawn : Spawns worker tasks.

pvmfinitsend : Gets the buffer ready to packed and then sent.

pvmfpack : Inserts the specified variable of the specified type into the buffer. An example would be the first time this appears in the program **par24-master.for**:

> **CALL pvmfpack(3, BIG_N(M), 1, 1, info)** - The first number tells what data type you are sending , 3 for integer or 6 for double precision real. Other possible values are given in the **PVM 3 USER'S GUIDE AND REFERENCE MANUAL**, [6]. The next value is the variable we are packing into the buffer; and then, we have the number that tells how big our variable is. In this example we are packing in **BIG_N(M)** which has only 1 position. Notice when **GAMMA** is packed into the buffer, this number is 3 since it is an array with 3 positions.

pvmfsend : Sends the buffer.

pvmfrecv : Receives the buffer.

pvmfunpack : Gathers the specified variable from the buffer. The values used in this are similar to those used in **pvmfpack**. If we look at the first time **par24-master.for** uses this we have the following function call:

> **CALL pvmfunpack(6, W1, MAXBIG_N, 1, info)** - Once again the first number tells what data type you are unpacking , 3 for integer or 6 for double precision real. As before, other possible values are given in the **PVM 3 USER'S GUIDE AND REFERENCE MANUAL**, [6]. The next value is the variable we are taking out of the buffer. Then, we have the number, or variable in this case, that tells how big our variable is. In this example we are unpacking **W1** which has **MAXBIG_N** position.

pvmfexit : Exits PVM.

pvmfparent : Obtains the id of the parent program which called the worker program.

Appendix B

Programs Used in This Thesis

The following are the programs that were used in this thesis and included in the appendix. Any program `*-doub.for` is not included since it is just the double precision version of the quadruple precision version which is furnished.

`par24-seq-quad.for` 34
`par24-master.for` 41
`par24-worker.for` 45
`hyp24-seq-quad.for` 52
`hyp46-seq-quad.for` 59
`hyp46-pf-quad.for` 69
`hyp-master.for` 78
`hyp-worker.for` 83

```
C     This program solves
C
C*********************************************************************C
      INTEGER  J, BIG_N(20), N, BIG_K(20), MU, I, NUM_TIMES, M
      REAL*16  RIGHT_SIDE(301), LEFT_SIDE(301,301), LOWER(301,301),
     -           UPPER(301,301), Z(301), W(301), U(301),
     -           TEMP(301), H, K, GAMMA(3), A(3), DELTA(3,3),
     -           A_0, NORM(20), TAU(3), ORDER
C
      OPEN (55,FILE='ThesisIn.dat',STATUS='OLD')
      READ (55,*)NUM_TIMES
      DO M = 1,NUM_TIMES
C
        READ (55,*)BIG_N(M),BIG_K(M)
        H = (1.0) / QFLOAT(BIG_N(M)+1)
        K = (1.0) / QFLOAT(BIG_K(M)+1)
C
        CALL Init ( GAMMA, A_0, A, DELTA, TAU)
        CALL Clear_Vector (U, BIG_N(M))
        MU = 3
C
        DO N = 1,BIG_K(M)
          CALL Clear_Vector (TEMP,BIG_N(M))
          DO J = 1,MU
            CALL Get_Right_Hand_Side (RIGHT_SIDE,U,N,BIG_N(M),J,MU,
     -                                K,H,A,DELTA,TAU)
            CALL Get_Left_Hand_Side (LEFT_SIDE,N,BIG_N(M),J,MU,K,H,
     -                               GAMMA)
            CALL Factor (LEFT_SIDE, LOWER, UPPER, BIG_N(M))
            CALL LSolve (LOWER, Z, RIGHT_SIDE, BIG_N(M))
            CALL USolve (UPPER, W, Z, BIG_N(M))
            DO I = 1,BIG_N(M)
              TEMP(I) = TEMP(I) + W(I)
            END DO
          END DO
C
          DO I = 1,BIG_N(M)
            U(I) = A_0*U(I) + TEMP(I)
          END DO
        END DO
C
        CALL Error (U,NORM(M),BIG_N(M),H)
      END DO
C
155   FORMAT (I6,I6,F)
156   FORMAT (I6,I6,F,F10.7)
      WRITE(60,155) BIG_N(1)+1,BIG_K(1)+1,NORM(1)
      DO M = 2,NUM_TIMES
        ORDER = ( QLOG( NORM(M-1)/NORM(M) ) / QLOG(QFLOAT(2)) )
        WRITE(60,156) BIG_N(M)+1,BIG_K(M)+1,NORM(M),ORDER
      END DO
C
      STOP
      END
C
C*********************************************************************C
```

```
      SUBROUTINE Init (GAMMA, A_0, A, DELTA, TAU)
C-----------------------------------------------------------------------C
C
      REAL*16 GAMMA(3), A_0, A(3), DELTA(3,3), TAU(3)
C
      A_0 = (-.6304261005497924)
      A(1) = (1454.094874359267)
      A(2) = (-3794.673554929712)
      A(3) = (2342.209106670995)
C
      GAMMA(1) = (1.06)
      GAMMA(2) = (1.07)
      GAMMA(3) = (1.075890333521764)
C
      DELTA(1,1) = (3567.278607850113)
      DELTA(1,2) = (-9621.288547664747)
      DELTA(1,3) = (6054.176606481301)
      DELTA(2,1) = (-7319.518083718724)
      DELTA(2,2) = (19811.01919385797)
      DELTA(2,3) = (-12490.83444347257)
      DELTA(3,1) = (5293.580042689434)
      DELTA(3,2) = (-14250.03134996801)
      DELTA(3,3) = (8956.61793945243)
C
      TAU(1) = 0.0
      TAU(2) = 0.5
      TAU(3) = 1.0
C
      RETURN
      END
C
C***********************************************************************C
      SUBROUTINE Get_Right_Hand_Side (R,U,N,BIG_N,J,MU,K,H,A,DELTA,TAU)
C-----------------------------------------------------------------------C
C
      REAL*16 R(301), U(301), K, H, A(3), DELTA(3,3), Tn, TEMP, F,
     -       TAU(3), T
      INTEGER N, J, BIG_N, MU, I, L
C
      Tn = QFLOAT(N)*K
      DO L = 1, BIG_N
        TEMP = (0.0)
        DO I = 1, MU
          T = Tn + TAU(I)*K
          CALL Get_F (F,L,T,H)
          TEMP = TEMP + DELTA(I,J) * F
        END DO
        R(L) = A(J)*U(L) + K*TEMP
      END DO
C
      RETURN
      END
C
C***********************************************************************C
      SUBROUTINE Get_F (F,I,T,H)
C-----------------------------------------------------------------------C
```

```
          DO I = 1, MU
            T = Tn + TAU(I)*K
            CALL Get_F (F,L,T,H)
            TEMP = TEMP + DELTA(I,J) * F
          END DO
          R(L) = A(J)*U(L) + K*TEMP
        END DO
C
        RETURN
        END
C
C*******************************************************************************C
        SUBROUTINE Get_F (F,I,T,H)
C-------------------------------------------------------------------------------C
C
        REAL*16 H, T, F, ALPHA, X
        INTEGER I
C
        ALPHA = (3.0)
        X = QFLOAT(I) * H
        F = (T**(ALPHA-(1.0))) * (ALPHA*X*((1.0)-X)+
     -      (2.0)*T)
C
        RETURN
        END
C
C*******************************************************************************C
        SUBROUTINE Get_Left_Hand_Side (L,N,BIG_N,J,MU,K,H,GAMMA)
C-------------------------------------------------------------------------------C
C
C       Calculates the left hand side of (1.1.4a) in the thesis.
C
C-------------------------------------------------------------------------------C
C
        REAL*16 L(301,301), H, K, GAMMA(3)
        INTEGER N, BIG_N, J, MU, ROW, COL
C
        CALL Clear_Matrix (L, BIG_N)
        ROW = 1
        L(ROW,ROW+1) = - GAMMA(J)*K*(H**(-2.0))
        L(ROW,ROW) = (1.0) + GAMMA(J)*K*2.0*(H**(-2.0))
        ROW = BIG_N
        L(ROW,ROW) = (1.0) + GAMMA(J)*K*2.0*(H**(-2.0))
        L(ROW,ROW-1) = - GAMMA(J)*K*(H**(-2.0))
        DO ROW = 2,BIG_N-1
          L(ROW,ROW+1) = - GAMMA(J)*K*(H**(-2.0))
          L(ROW,ROW) = (1.0) + GAMMA(J)*K*2.0*(H**(-2.0))
          L(ROW,ROW-1) = - GAMMA(J)*K*(H**(-2.0))
        END DO
C
        RETURN
        END
C
C*******************************************************************************C
        SUBROUTINE Clear_Matrix (A, N)
```

```
      SUBROUTINE CLEAR_VECTOR (A, N)
C-----------------------------------------------------------------------C
C                                                                       C
C     This procedure sets all entries in vector A to (0.0).          C
C                                                                       C
C-----------------------------------------------------------------------C
C
      REAL*16 A(301)
      INTEGER ROW, N
C
      DO ROW = 1,N
          A(ROW) = (0.0)
      END DO
      RETURN
      END
C
C***********************************************************************C
      SUBROUTINE FACTOR(A, L, U, N)
C-----------------------------------------------------------------------C
C                                                                       C
C     This procedure factors A into L and U where L is a lower          C
C     triangular matrix and U is an upper triangular matrix.            C
C                                                                       C
C-----------------------------------------------------------------------C
C
      REAL*16 TEMP, A(301,301), L(301,301), U(301,301)
      INTEGER I, J, K, N
C
      DO I = 1,N
        L(I,I) = (1.0)
      END DO
C
      U(1,1) = A(1,1)
      IF (U(1,1).EQ.(0.0)) THEN
1100     FORMAT(' 1  FACTORIZATION IMPOSSIBLE!')
1101     FORMAT(' 2  FACTORIZATION IMPOSSIBLE!')
        WRITE (6,1100)
        STOP
      END IF
C
      DO I = 2,N
        U(1,I) = A(1,I) / L(1,1)
        L(I,1) = A(I,1) / U(1,1)
      END DO
C
      DO I = 2,N-1
        TEMP = (0.0)
        DO K = 1,I-1
          TEMP = TEMP + (L(I,K)*U(K,I))
        END DO
        U(I,I) = A(I,I) - TEMP
        IF (U(I,I).EQ.(0.0)) THEN
          WRITE (6,1101)
          STOP
        END IF
C
```

```
        DO J = I+1,N
          TEMP = (0.0)
          DO K = 1,I-1
            TEMP = TEMP + (L(I,K) * U(K,J))
          END DO
          U(I,J) = (A(I,J) - TEMP) / L(I,I)
C
          TEMP = (0.0)
          DO K = 1,I-1
            TEMP = TEMP + (L(J,K) * U(K,I))
          END DO
          L(J,I) = (A(J,I) - TEMP) / U(I,I)
        END DO
      END DO
C
      TEMP = (0.0)
      dO K = 1,N-1
        TEMP = TEMP + (L(N,K) * U(K,N))
      END DO
      U(N,N) = A(N,N) - TEMP
      RETURN
      END
C
C*********************************************************************C
      SUBROUTINE TRANSPOSE (A,TRANS,N)
C---------------------------------------------------------------------C
C                                                                     C
C     This procedure figures the transpose of matrix A and puts the   C
C     answer in TRANS.                                                C
C                                                                     C
C---------------------------------------------------------------------C
C
      REAL*16 A(301,301), TRANS(301,301)
      INTEGER N, ROW, COL
C
      DO ROW = 1,N
        DO COL = 1,N
          TRANS(COL,ROW) = A(ROW,COL)
        END DO
      END DO
      RETURN
      END
C
C*********************************************************************C
      SUBROUTINE USOLVE(U, W, X, N)
C---------------------------------------------------------------------C
C                                                                     C
C     This procedure takes an upper triangular matrix, U, and solves  C
C     the equation UW=X for W.                                        C
C                                                                     C
C---------------------------------------------------------------------C
C
      REAL*16 U(301,301), W(301), X(301), TEMP
      INTEGER I, J, K, N
C
      W(N) = X(N) / U(N,N)
```

```
        END DO
      END DO
      RETURN
      END
C
C*************************************************************************C
      SUBROUTINE USOLVE(U, W, X, N)
C-------------------------------------------------------------------------C
C                                                                         C
C     This procedure takes an upper triangular matrix, U, and solves    C
C     the equation UW=X for W.                                           C
C                                                                         C
C-------------------------------------------------------------------------C
C
      REAL*16 U(301,301), W(301), X(301), TEMP
      INTEGER I, J, K, N
C
      W(N) = X(N) / U(N,N)
C
      DO I = N - 1,1,-1
        DO J = 1,N
          TEMP = (0.0)
          DO K = I + 1,N
            TEMP = TEMP + (U(I,K)*W(K))
          END DO
          W(I) = (X(I) - TEMP) / U(I,I)
        END DO
      END DO
      RETURN
      END
C
C*************************************************************************C
      SUBROUTINE LSOLVE(L, M, W, N)
C-------------------------------------------------------------------------C
C                                                                         C
C     This procedure takes a lower triangular matrix, L, and solves the C
C     equation LM=W for M.                                               C
C                                                                         C
C-------------------------------------------------------------------------C
C
      REAL*16 L(301,301), M(301), W(301), TEMP
      INTEGER I, J, K, N
C
      DO I = 1,N
        M(1) = W(1)
      END DO
C
      DO I = 2,N
        DO J = 1,N
          TEMP = (0.0)
          DO K = 1,I - 1
            TEMP = TEMP + (L(I,K)*M(K))
          END DO
          M(I) = (W(I) - TEMP)
        END DO
      END DO
```

```
      END DO
      RETURN
      END
C
C*********************************************************************C
      SUBROUTINE Error (U,NORM,N,H)
C-----------------------------------------------------------------------C
C                                                                       C
C     This procedure figures the error between U and U_EXACT.            C
C                                                                       C
C-----------------------------------------------------------------------C
C
      REAL*16 U(301), U_EXACT, NORM, TEMP, H, T
      INTEGER N, I
C
      NORM = (0.0)
      T = (1.0)
      DO I = 1,N
        CALL Get_U_Exact (U_EXACT, (QFLOAT(I)*H), T)
1402    FORMAT (F36.33,I6,F37.33)
c        WRITE (6,1402)I*H,I,U_EXACT
        TEMP = QABS(U(I) - U_EXACT)
        IF (TEMP.GT.NORM) NORM = TEMP
      END DO
      RETURN
      END
C*********************************************************************C
      SUBROUTINE Get_U_Exact (U,X,T)
C-----------------------------------------------------------------------C
C                                                                       C
C     This subroutine returns the value of u(x,t).                       C
C                                                                       C
C-----------------------------------------------------------------------C
C
      REAL*16 X, U, T, ALPHA
C
      ALPHA = 3.0
      U = T**ALPHA * X * ((1.0) - X)
      RETURN
      END
```

```
      PROGRAM master
C*************************************************************************C
C
C     par24-Master.for
C
C     This program solves an parabolic PDE using PVM.  You need the
C     program WORKER too.
C
C*************************************************************************C
      INCLUDE 'pvm3/include/fpvm3.h'
C
      INTEGER  J, BIG_N(20), N, BIG_K(20), MU, I, MAXBIG_N,
     -         NUM_TIMES, M
      INTEGER NPROCS
      PARAMETER (NPROCS = 3, MU = 3, MAXBIG_N = 301)
      INTEGER FLAG, mytid, task_ids(NPROCS), numt, info, msgtype
      REAL*8  W1(MAXBIG_N), W2(MAXBIG_N), W3(MAXBIG_N),
     -        U(MAXBIG_N), H, K, GAMMA(3), A(3), DELTA(3,3),
     -        A_0, NORM(20), TAU(3), ORDER
C
C     * Enroll in PVM *
      CALL pvmfmytid(mytid)
C
      OPEN (55,FILE='ThesisIn.dat',STATUS='OLD')
      READ (55,*)NUM_TIMES
      DO M = 1,NUM_TIMES
C
        READ (55,*)BIG_N(M),BIG_K(M)
        H = (1.0) / DFLOAT(BIG_N(M)+1)
        K = (1.0) / DFLOAT(BIG_K(M)+1)
C
        CALL Init ( GAMMA, A_0, A, DELTA, TAU )
        CALL Clear_Vector (U, BIG_N(M))
C
        DO N = 1,BIG_K(M)
          FLAG = PVMDEFAULT
C
C     * Spawn worker tasks *
          CALL pvmfspawn('ell2-worker', FLAG, '', NPROCS,
     -                   task_ids, numt)
          msgtype = 1

          DO J = 1,MU
            CALL pvmfinitsend(PVMDEFAULT, info)
            CALL pvmfpack(3, BIG_N(M), 1, 1, info)
            CALL pvmfpack(6, U, MAXBIG_N, 1, info)
            CALL pvmfpack(6, K, 1, 1, info)
            CALL pvmfpack(6, H, 1, 1, info)
            CALL pvmfpack(6, A, 3, 1, info)
            CALL pvmfpack(6, DELTA, 9, 1, info)
            CALL pvmfpack(6, TAU, 3, 1, info)
            CALL pvmfpack(6, GAMMA, 3, 1, info)
            CALL pvmfpack(3, N, 1, 1, info)
            CALL pvmfpack(3, J, 1, 1, info)
            CALL pvmfpack(3, MU, 1, 1, info)
            CALL pvmfsend (task_ids(j), msgtype, info)
```

```
          END DO
C
C       * Get results *
          msgtype = 7
          CALL pvmfrecv(task_ids(1), msgtype, info)
          CALL pvmfunpack(6, W1, MAXBIG_N, 1, info)
C
          CALL pvmfrecv(task_ids(2), msgtype, info)
          CALL pvmfunpack(6, W2, MAXBIG_N, 1, info)
C
          CALL pvmfrecv(task_ids(3), msgtype, info)
          CALL pvmfunpack(6, W3, MAXBIG_N, 1, info)
C
          DO i = 1,BIG_N(M)
            U(I) = A_0*U(I) + W1(I) + W2(I) + W3(I)
          END DO

        END DO
C
        CALL Error (U,NORM(M),BIG_N(M),H)
      END DO
C
155   FORMAT (I6,I6,F)
156   FORMAT (I6,I6,F,F10.7)
      WRITE(65,155) BIG_N(1)+1,BIG_K(1)+1,NORM(1)
      DO M = 2,NUM_TIMES
        ORDER = ( DLOG( NORM(M-1)/NORM(M) ) / DLOG(DFLOAT(2)) )
        WRITE(65,156) BIG_N(M)+1,BIG_K(M)+1,NORM(M),ORDER
      END DO
C
      CALL pvmfexit(info)
      STOP
      END
C
C*************************************************************************C
      SUBROUTINE Init (GAMMA, A_0, A, DELTA, TAU)
C-------------------------------------------------------------------------C
C
C     This subroutine initializes GAMMA, A_0, A, DELTA and TAU.
C
C-------------------------------------------------------------------------C
C
      REAL*8 GAMMA(3), A_0, A(3), DELTA(3,3), TAU(3)
C
      A_0 = (-.6304261005497924)
      A(1) = (1454.094874359267)
      A(2) = (-3794.673554929712)
      A(3) = (2342.209106670995)
C
      GAMMA(1) = (1.06)
      GAMMA(2) = (1.07)
      GAMMA(3) = (1.075890333521764)
C
      DELTA(1,1) = (3567.278607850113)
      DELTA(1,2) = (-9621.288547664747)
      DELTA(1,3) = (6054.176606481301)
```

```
      DELTA(2,1) = (-7319.518083718724)
      DELTA(2,2) = (19811.01919385797)
      DELTA(2,3) = (-12490.83444347257)
      DELTA(3,1) = (5293.580042689434)
      DELTA(3,2) = (-14250.03134996801)
      DELTA(3,3) = (8956.617973945243)
C
      TAU(1) = 0.0
      TAU(2) = 0.5
      TAU(3) = 1.0
C
      RETURN
      END
C
C
C*************************************************************************C
      SUBROUTINE CLEAR_VECTOR (A, N)
C-------------------------------------------------------------------------C
C                                                                         C
C     This procedure sets all entries in vector A to (0.0).               C
C                                                                         C
C-------------------------------------------------------------------------C
C
      INTEGER MAXBIG_N
      PARAMETER (MAXBIG_N = 301)
      REAL*8 A(MAXBIG_N)
      INTEGER ROW, N
C
      DO ROW = 1,N
          A(ROW) = (0.0)
      END DO
      RETURN
      END
C
C
C*************************************************************************C
      SUBROUTINE Error (U,NORM,N,H)
C-------------------------------------------------------------------------C
C                                                                         C
C     This procedure figures the error between U and U_EXACT.             C
C                                                                         C
C-------------------------------------------------------------------------C
C
      INTEGER MAXBIG_N
      PARAMETER (MAXBIG_N = 301)
      REAL*8 U(MAXBIG_N), U_EXACT, NORM, TEMP, H, T
      INTEGER N, I
C
      NORM = (0.0)
      T = (1.0)
      NORM = 0.0
C      WRITE (60,*) ('U_EXACT IS')
      DO I = 1,N
        CALL Get_U_Exact (U_EXACT, (DFLOAT(I)*H), T)
1402    FORMAT (F36.33,I6,F37.33)
C        WRITE (60,1402)I*H,I,U_EXACT
```

```
          TEMP = DABS(U(I) - U_EXACT)
          IF (TEMP.GT.NORM) NORM = TEMP
        END DO
        RETURN
        END

C*************************************************************************C
        SUBROUTINE Get_U_Exact (U,X,T)
C-------------------------------------------------------------------------C
C                                                                         C
C       This subroutine returns the value of u(x,t).                      C
C                                                                         C
C-------------------------------------------------------------------------C
C
        REAL*8 X, U, T, ALPHA
C
        ALPHA = 3.0
        U = T**ALPHA * X * ((1.0) - X)
        RETURN
        END
```

```
      PROGRAM worker
C*************************************************************************C
C
C     Program par24-Worker.for
C
C     This program is the worker program.  It helps MASTER solve an
C     parabolic PDE.
C
C*************************************************************************C
C
      INCLUDE 'pvm3/include/fpvm3.h'
C
      INTEGER  J, BIG_N, N, MU, MAXBIG_N
      PARAMETER ( MAXBIG_N = 301)
      INTEGER mytid, info, mastertid, msgtype
      REAL*8  W(MAXBIG_N), U(MAXBIG_N),
     -         H, K, GAMMA(3), A(3), DELTA(3,3),
     -         TAU(3),
     -        RIGHT_SIDE(MAXBIG_N), LEFT_SIDE(MAXBIG_N,MAXBIG_N),
     -        LOWER(MAXBIG_N,MAXBIG_N), UPPER(MAXBIG_N,MAXBIG_N),
     -        Z(MAXBIG_N)
C
C     * Enroll in PVM *
      CALL pvmfmytid( mytid )
C     * Get the master's task id *
      CALL pvmfparent( mastertid )
C
C     * Receive info needed from master *
      msgtype = 1
      CALL pvmfrecv( mastertid, msgtype, info )
      CALL pvmfunpack( 3, BIG_N, 1, 1, info )
      CALL pvmfunpack( 6, U, MAXBIG_N, 1, info )
      CALL pvmfunpack( 6, K, 1, 1, info )
      CALL pvmfunpack( 6, H, 1, 1, info )
      CALL pvmfunpack( 6, A, 3, 1, info )
      CALL pvmfunpack( 6, DELTA, 9, 1, info )
      CALL pvmfunpack( 6, TAU, 3, 1, info )
      CALL pvmfunpack( 6, GAMMA, 3, 1, info )
      CALL pvmfunpack( 3, N, 1, 1, info )
      CALL pvmfunpack( 3, J, 1, 1, info )
      CALL pvmfunpack( 3, MU, 1, 1, info )

C       * Work involved *
      CALL Get_Right_Hand_Side (RIGHT_SIDE,U,N,BIG_N,J,MU,K,H,A,
     -                          DELTA,TAU)
      CALL Get_Left_Hand_Side (LEFT_SIDE,N,BIG_N,J,MU,K,H,GAMMA)
      CALL Factor (LEFT_SIDE, LOWER, UPPER, BIG_N)
      CALL LSolve (LOWER, Z, RIGHT_SIDE, BIG_N)
      CALL USolve (UPPER, W, Z, BIG_N)
C
C
C       * Send back results *
      msgtype = 7
      CALL pvmfinitsend( PVMDEFAULT, info )
      CALL pvmfpack( 6, W, MAXBIG_N, 1, info )
      CALL pvmfsend( mastertid, msgtype, info )
```

```
      CALL pvmfexit()
      STOP
      END
C
C*************************************************************************C
      SUBROUTINE Get_Right_Hand_Side (R,U,N,BIG_N,J,MU,K,H,A,DELTA,TAU)
C-------------------------------------------------------------------------C
C
C     This subroutine calculates the right hand side in (1.1.4a) of the
C     thesis.
C
C-------------------------------------------------------------------------C
C
      INTEGER MAXBIG_N
      PARAMETER (MAXBIG_N = 301)
      REAL*8 R(MAXBIG_N), U(MAXBIG_N), K, H, A(3), DELTA(3,3), Tn,
     -       TEMP, F, TAU(3), T
      INTEGER N, J, BIG_N, MU, I, L
C
      Tn = DFLOAT(N)*K
      DO L = 1, BIG_N
        TEMP = (0.0)
        DO I = 1, MU
          T = Tn + TAU(I)*K
          CALL Get_F (F,L,T,H)
          TEMP = TEMP + DELTA(I,J) * F
        END DO
        R(L) = A(J)*U(L) + K*TEMP
      END DO
C
      RETURN
      END
C
C*************************************************************************C
      SUBROUTINE Get_F (F,I,T,H)
C-------------------------------------------------------------------------C
C
      REAL*8 H, T, F, ALPHA, X
      INTEGER I
C
      ALPHA = (3.0)
      X = DFLOAT(I) * H
      F = (T**(ALPHA-(1.0))) * (ALPHA*X*((1.0)-X)+
     -    (2.0)*T)
C
      RETURN
      END
C
C*************************************************************************C
      SUBROUTINE Get_Left_Hand_Side (L,N,BIG_N,J,MU,K,H,GAMMA)
C-------------------------------------------------------------------------C
C
C     This subroutine calculates the left hand side in (1.1.4a) of the
C     thesis.
C
```

```
C-----------------------------------------------------------------------C
C
      INTEGER MAXBIG_N
      PARAMETER (MAXBIG_N = 301)
      REAL*8 L(MAXBIG_N,MAXBIG_N), H, K, GAMMA(3)
      INTEGER N, BIG_N, J, MU, ROW
C
      CALL Clear_Matrix (L, BIG_N)
      ROW = 1
      L(ROW,ROW+1) = - GAMMA(J)*K*(H**(-2.0))
      L(ROW,ROW) = (1.0) + GAMMA(J)*K*2.0*(H**(-2.0))
      ROW = BIG_N
      L(ROW,ROW) = (1.0) + GAMMA(J)*K*2.0*(H**(-2.0))
      L(ROW,ROW-1) = - GAMMA(J)*K*(H**(-2.0))
      DO ROW = 2,BIG_N-1
        L(ROW,ROW+1) = - GAMMA(J)*K*(H**(-2.0))
        L(ROW,ROW) = (1.0) + GAMMA(J)*K*2.0*(H**(-2.0))
        L(ROW,ROW-1) = - GAMMA(J)*K*(H**(-2.0))
      END DO
C
      RETURN
      END
C
C***********************************************************************C
      SUBROUTINE Clear_Matrix (A, N)
C-----------------------------------------------------------------------C
C                                                                       C
C     This procedure sets all entries in matrix A to (0.0).             C
C                                                                       C
C-----------------------------------------------------------------------C
C
      INTEGER MAXBIG_N
      PARAMETER (MAXBIG_N = 301)
      REAL*8 A(MAXBIG_N,MAXBIG_N)
      INTEGER ROW, COL, N
C
      DO ROW = 1,N
        DO COL = 1,N
          A(ROW,COL) = (0.0)
        END DO
      END DO
      RETURN
      END
C
C***********************************************************************C
      SUBROUTINE CLEAR_VECTOR (A, N)
C-----------------------------------------------------------------------C
C                                                                       C
C     This procedure sets all entries in vector A to (0.0).             C
C                                                                       C
C-----------------------------------------------------------------------C
C
      INTEGER MAXBIG_N
      PARAMETER (MAXBIG_N = 301)
      REAL*8 A(MAXBIG_N)
```

```
      INTEGER ROW, N
C
      DO ROW = 1,N
          A(ROW) = (0.0)
      END DO
      RETURN
      END
C
C*************************************************************************C
      SUBROUTINE FACTOR(A, L, U, N)
C-------------------------------------------------------------------------C
C                                                                         C
C     This procedure factors A into L and U where L is a lower            C
C     triangular matrix and U is an upper triangular matrix.              C
C                                                                         C
C-------------------------------------------------------------------------C
C
      INTEGER MAXBIG_N
      PARAMETER (MAXBIG_N = 301)
      REAL*8 TEMP, A(MAXBIG_N,MAXBIG_N), L(MAXBIG_N,MAXBIG_N),
     -       U(MAXBIG_N,MAXBIG_N)
      INTEGER I, J, K, N
C
      DO I = 1,N
        L(I,I) = (1.0)
      END DO
C
      U(1,1) = A(1,1)
      IF (U(1,1).EQ.(0.0)) THEN
1100     FORMAT(' 1  FACTORIZATION IMPOSSIBLE!')
1101     FORMAT(' 2  FACTORIZATION IMPOSSIBLE!')
        WRITE (6,1100)
        STOP
      END IF
C
      DO I = 2,N
        U(1,I) = A(1,I) / L(1,1)
        L(I,1) = A(I,1) / U(1,1)
      END DO
C
      DO I = 2,N-1
        TEMP = (0.0)
        DO K = 1,I-1
          TEMP = TEMP + (L(I,K)*U(K,I))
        END DO
        U(I,I) = A(I,I) - TEMP
        IF (U(I,I).EQ.(0.0)) THEN
          WRITE (6,1101)
          STOP
        END IF
C
        DO J = I+1,N
          TEMP = (0.0)
          DO K = 1,I-1
            TEMP = TEMP + (L(I,K) * U(K,J))
          END DO
```

```
          U(I,J) = (A(I,J) - TEMP) / L(I,I)
C
          TEMP = (0.0)
          DO K = 1,I-1
            TEMP = TEMP + (L(J,K) * U(K,I))
          END DO
          L(J,I) = (A(J,I) - TEMP) / U(I,I)
        END DO
      END DO
C
      TEMP = (0.0)
      dO K = 1,N-1
        TEMP = TEMP + (L(N,K) * U(K,N))
      END DO
      U(N,N) = A(N,N) - TEMP
      RETURN
      END
C
C*********************************************************************C
      SUBROUTINE TRANSPOSE (A,TRANS,N)
C---------------------------------------------------------------------C
C                                                                     C
C     This procedure figures the transpose of matrix A and puts the   C
C     answer in TRANS.                                                C
C                                                                     C
C---------------------------------------------------------------------C
C
      INTEGER MAXBIG_N
      PARAMETER (MAXBIG_N = 301)
      REAL*8 A(MAXBIG_N,MAXBIG_N), TRANS(MAXBIG_N,MAXBIG_N)
      INTEGER N, ROW, COL
C
      DO ROW = 1,N
        DO COL = 1,N
          TRANS(COL,ROW) = A(ROW,COL)
        END DO
      END DO
      RETURN
      END
C
C*********************************************************************C
      SUBROUTINE USOLVE(U, W, X, N)
C---------------------------------------------------------------------C
C                                                                     C
C     This procedure takes an upper triangular matrix, U, and solves  C
C     the equation UW=X for W.                                        C
C                                                                     C
C---------------------------------------------------------------------C
C
      INTEGER MAXBIG_N
      PARAMETER (MAXBIG_N = 301)
      REAL*8 U(MAXBIG_N,MAXBIG_N), W(MAXBIG_N), X(MAXBIG_N), TEMP
      INTEGER I, J, K, N
C
      W(N) = X(N) / U(N,N)
C
```

```
      DO I = N - 1,1,-1
        DO J = 1,N
          TEMP = (0.0)
          DO K = I + 1,N
            TEMP = TEMP + (U(I,K)*W(K))
          END DO
          W(I) = (X(I) - TEMP) / U(I,I)
        END DO
      END DO
      RETURN
      END
C
C*********************************************************************C
      SUBROUTINE LSOLVE(L, M, W, N)
C---------------------------------------------------------------------C
C                                                                     C
C     This procedure takes a lower triangular matrix, L, and solves the C
C     equation LM=W for M.                                            C
C                                                                     C
C---------------------------------------------------------------------C
C
      INTEGER MAXBIG_N
      PARAMETER (MAXBIG_N = 301)
      REAL*8 L(MAXBIG_N,MAXBIG_N), M(MAXBIG_N), W(MAXBIG_N), TEMP
      INTEGER I, J, K, N
C
      DO I = 1,N
        M(1) = W(1)
      END DO
C
      DO I = 2,N
        DO J = 1,N
          TEMP = (0.0)
          DO K = 1,I - 1
            TEMP = TEMP + (L(I,K)*M(K))
          END DO
          M(I) = (W(I) - TEMP)
        END DO
      END DO
      RETURN
      END
C
C*********************************************************************C
      SUBROUTINE ADD (A, B, N)
C---------------------------------------------------------------------C
C                                                                     C
C     This procedure does A+B=A where A and B are NxN matrices.       C
C                                                                     C
C---------------------------------------------------------------------C
C
      INTEGER MAXBIG_N
      PARAMETER (MAXBIG_N = 301)
      REAL*8 A(MAXBIG_N,MAXBIG_N), B(MAXBIG_N,MAXBIG_N)
      INTEGER ROW, COL, N
C
      DO ROW = 1,N
```

```
          DO COL = 1,N
            A(ROW,COL) = A(ROW,COL) + B(ROW,COL)
          END DO
        END DO
        RETURN
        END
C
```

```
C*****************************************************************C
C
C     Hyp24-seq-quad.for
C
C     This program solves the hyperbolic PDE using a general 2nd
C     order scheme which is 4th order for certain constants.
C
C*****************************************************************C
      INTEGER  ROW, COL, J, BIG_N, N, BIG_K, I
      REAL*16  RIGHT_SIDE(350), LEFT_SIDE(350,350), LOWER(350,350),
     -           UPPER(350,350), Z(350), W(350), U(350), U1(350),
     -           U2(350), H, K, BETA, THETA, NORM
C
C
      READ (5,*)BIG_N
      H = (1.0) / QFLOAT(BIG_N+1)
      READ (5,*)BIG_K
      K = (1.0) / QFLOAT(BIG_K+1)
C
      CALL Init (BETA, THETA, U, U1, H, K, BIG_N)
102   FORMAT ('  N = ',I8,'  K = ',I8,'  Beta = ',F10.8,
     -        '  Theta = ',F10.8)
      WRITE (6,102) BIG_N,BIG_K,BETA,THETA
C
        CALL Get_Left_Hand_Side (LEFT_SIDE,BIG_N,K,H,BETA)
C
      DO N = 1,BIG_K
        CALL Get_Right_Hand_Side (RIGHT_SIDE,U1,N,BIG_N,K,H,THETA)

        CALL Factor (LEFT_SIDE, LOWER, UPPER, BIG_N)
        CALL LSolve (LOWER, Z, RIGHT_SIDE, BIG_N)
        CALL USolve (UPPER, W, Z, BIG_N)
C
        DO I = 1,BIG_N
          U2(I) = W(I) + 2*U1(I) - U(I)
          U(I) = U1(I)
          U1(I) = U2(I)
        END DO

      END DO
C
      CALL Error (U2,NORM,BIG_N,H)
      WRITE (6,*) ('THE ERROR IS ')
      WRITE (6,*) NORM
      Write (6,*)

      STOP
      END
C
C*****************************************************************C
      SUBROUTINE Init (BETA, THETA, U, U1, H, K, BIG_N)
C-----------------------------------------------------------------C
C
      REAL*16 BETA, THETA, U(350), U1(350), H, K
      INTEGER BIG_N, I
C
```

```
c       BETA = 1.0
C       THETA = 1.0
        BETA = QFLOAT(1)/QFLOAT(12)
        THETA = QFLOAT(1)/QSQRT(QFLOAT(6))
c       WRITE (6,*) ('Enter BETA')
c       READ (5,*)BETA
c       WRITE (6,*) ('Enter THETA')
c       READ (5,*)THETA

        CALL Clear_Vector (U, BIG_N)
        DO I = 1,BIG_N
          CALL Get_U_Exact (U1(I), (QFLOAT(I)*H), K)
        END DO
C
        RETURN
        END
C
C*******************************************************************************C
        SUBROUTINE Get_Right_Hand_Side (R,U1,N,BIG_N,K,H,THETA)
C-------------------------------------------------------------------------------C
C
C
C       This subroutine figures out the right hand side of (2.1.7a).
C
C-------------------------------------------------------------------------------C
C
        REAL*16 R(350), U1(350), K, H, THETA, Tn, TEMP, F1, F2, T1, T2
        INTEGER N, BIG_N, I, L
C
        Tn = QFLOAT(N)*K
C
        DO L = 1, BIG_N
C
          IF (L.EQ.1) THEN
            TEMP = -((K/H)**2) * (2*U1(L) - U1(L+1))
          ELSE IF (L.EQ.BIG_N) THEN
            TEMP = -((K/H)**2) * (-U1(L-1) + 2*U1(L))
          ELSE
            TEMP = -((K/H)**2) * (-U1(L-1) + 2*U1(L) - U1(L+1))
          END IF
C
          T1 = Tn + THETA*K
          T2 = Tn - THETA*K
          CALL Get_F (F1,L,T1,H)
          CALL Get_F (F2,L,T2,H)
          R(L) = TEMP + .5*(K**2)*(F1 + F2)
        END DO
C
        RETURN
        END
C
C*******************************************************************************C
        SUBROUTINE Get_F (F,I,T,H)
C-------------------------------------------------------------------------------C
C
        REAL*16 H, T, F, ALPHA, X
```

```
      INTEGER I
C
      ALPHA = (4.0)
      X = QFLOAT(I) * H
      F = (T**(ALPHA-(2.0))) * (ALPHA*(ALPHA-1.0)*X*((1.0)-X)+
     -    (2.0)*(T**2))
C
      RETURN
      END
C
C*********************************************************************C
      SUBROUTINE Get_Left_Hand_Side (L,BIG_N,K,H,BETA)
C---------------------------------------------------------------------C
C
C
C     This subroutine figures out the left hand side of (2.1.7a).
C
C---------------------------------------------------------------------C
C
      REAL*16 L(350,350), H, K, BETA
      INTEGER  BIG_N, ROW
C
      CALL Clear_Matrix (L, BIG_N)
      IF (BIG_N.EQ.1) THEN
        ROW = 1
        L(ROW,ROW+1) = - BETA*(K**2)*(H**(-2.0))
        L(ROW,ROW) = (1.0) + BETA*(K**2)*2.0*(H**(-2.0))
      ELSE
        ROW = 1
        L(ROW,ROW+1) = - BETA*(K**2)*(H**(-2.0))
        L(ROW,ROW) = (1.0) + BETA*(K**2)*2.0*(H**(-2.0))
        ROW = BIG_N
        L(ROW,ROW) = (1.0) + BETA*(K**2)*2.0*(H**(-2.0))
        L(ROW,ROW-1) = - BETA*(K**2)*(H**(-2.0))
        DO ROW = 2,BIG_N-1
          L(ROW,ROW+1) = - BETA*(K**2)*(H**(-2.0))
          L(ROW,ROW) = (1.0) + BETA*(K**2)*2.0*(H**(-2.0))
          L(ROW,ROW-1) = - BETA*(K**2)*(H**(-2.0))
        END DO
      END IF
C
      RETURN
      END
C
C*********************************************************************C
      SUBROUTINE Clear_Matrix (A, N)
C---------------------------------------------------------------------C
C                                                                     C
C     This procedure sets all entries in matrix A to (0.0).         C
C                                                                     C
C---------------------------------------------------------------------C
C
      REAL*16 A(350,350)
      INTEGER ROW, COL, N
C
```

```
      DO ROW = 1,N
        DO COL = 1,N
          A(ROW,COL) = (0.0)
        END DO
      END DO
      RETURN
      END
C
C*************************************************************************C
      SUBROUTINE CLEAR_VECTOR (A, N)
C-------------------------------------------------------------------------C
C                                                                         C
C     This procedure sets all entries in vector A to (0.0).         C
C                                                                         C
C-------------------------------------------------------------------------C
C
      REAL*16 A(350)
      INTEGER ROW, N
C
      DO ROW = 1,N
          A(ROW) = (0.0)
      END DO
      RETURN
      END
C
C*************************************************************************C
      SUBROUTINE FACTOR(A, L, U, N)
C-------------------------------------------------------------------------C
C                                                                         C
C     This procedure factors A into L and U where L is a lower          C
C     triangular matrix and U is an upper triangular matrix.            C
C                                                                         C
C-------------------------------------------------------------------------C
C
      REAL*16 TEMP, A(350,350), L(350,350), U(350,350)
      INTEGER I, J, K, N
C
      DO I = 1,N
        L(I,I) = (1.0)
      END DO
C
      U(1,1) = A(1,1)
      IF (U(1,1).EQ.(0.0)) THEN
1100     FORMAT(' 1  FACTORIZATION IMPOSSIBLE!')
1101     FORMAT(' 2  FACTORIZATION IMPOSSIBLE!')
        WRITE (6,1100)
        STOP
      END IF
C
      DO I = 2,N
        U(1,I) = A(1,I) / L(1,1)
        L(I,1) = A(I,1) / U(1,1)
      END DO
C
      DO I = 2,N-1
        TEMP = (0.0)
```

```
        DO K = 1,I-1
          TEMP = TEMP + (L(I,K)*U(K,I))
        END DO
        U(I,I) = A(I,I) - TEMP
        IF (U(I,I).EQ.(0.0)) THEN
          WRITE (6,1101)
          STOP
        END IF
C
        DO J = I+1,N
          TEMP = (0.0)
          DO K = 1,I-1
            TEMP = TEMP + (L(I,K) * U(K,J))
          END DO
          U(I,J) = (A(I,J) - TEMP) / L(I,I)
C
          TEMP = (0.0)
          DO K = 1,I-1
            TEMP = TEMP + (L(J,K) * U(K,I))
          END DO
          L(J,I) = (A(J,I) - TEMP) / U(I,I)
        END DO
      END DO
C
      TEMP = (0.0)
      dO K = 1,N-1
        TEMP = TEMP + (L(N,K) * U(K,N))
      END DO
      U(N,N) = A(N,N) - TEMP
      RETURN
      END
C
C*======================================================================C
      SUBROUTINE TRANSPOSE (A,TRANS,N)
C----------------------------------------------------------------------C
C                                                                      C
C     This procedure figures the transpose of matrix A and puts the    C
C     answer in TRANS.                                                 C
C                                                                      C
C----------------------------------------------------------------------C
C
      REAL*16 A(350,350), TRANS(350,350)
      INTEGER N, ROW, COL
C
      DO ROW = 1,N
        DO COL = 1,N
          TRANS(COL,ROW) = A(ROW,COL)
        END DO
      END DO
      RETURN
      END
C
C*======================================================================C
      SUBROUTINE USOLVE(U, W, X, N)
C----------------------------------------------------------------------C
C                                                                      C
```

```
C     This procedure takes an upper triangular matrix, U, and solves    C
C     the equation UW=X for W.                                           C
C                                                                        C
C------------------------------------------------------------------------C
C
      REAL*16 U(350,350), W(350), X(350), TEMP
      INTEGER I, J, K, N
C
      W(N) = X(N) / U(N,N)
C
      DO I = N - 1,1,-1
        DO J = 1,N
          TEMP = (0.0)
          DO K = I + 1,N
            TEMP = TEMP + (U(I,K)*W(K))
          END DO
          W(I) = (X(I) - TEMP) / U(I,I)
        END DO
      END DO
      RETURN
      END
C
C************************************************************************C
      SUBROUTINE LSOLVE(L, M, W, N)
C------------------------------------------------------------------------C
C                                                                        C
C     This procedure takes a lower triangular matrix, L, and solves the C
C     equation LM=W for M.                                               C
C                                                                        C
C------------------------------------------------------------------------C
C
      REAL*16 L(350,350), M(350), W(350), TEMP
      INTEGER I, J, K, N
C
      DO I = 1,N
        M(1) = W(1)
      END DO
C
      DO I = 2,N
        DO J = 1,N
          TEMP = (0.0)
          DO K = 1,I - 1
            TEMP = TEMP + (L(I,K)*M(K))
          END DO
          M(I) = (W(I) - TEMP)
        END DO
      END DO
      RETURN
      END
C
C************************************************************************C
      SUBROUTINE ADD (A, B, N)
C------------------------------------------------------------------------C
C                                                                        C
C     This procedure does A+B=A where A and B are NxN matrices.          C
C                                                                        C
```

```
C-----------------------------------------------------------------------C
C
      REAL*16 A(350,350), B(350,350)
      INTEGER ROW, COL, N
C
      DO ROW = 1,N
        DO COL = 1,N
          A(ROW,COL) = A(ROW,COL) + B(ROW,COL)
        END DO
      END DO
      RETURN
      END
C
C*********************************************************************C
      SUBROUTINE Error (U,NORM,N,H)
C-----------------------------------------------------------------------C
C                                                                       C
C     This procedure figures the error between U and U_EXACT.            C
C                                                                       C
C-----------------------------------------------------------------------C
C
      REAL*16 U(350), U_EXACT, NORM, TEMP, H, T
      INTEGER N, I
C
      NORM = (0.0)
      T = (1.0)
      NORM = 0.0
c      WRITE (70,*) ('U_EXACT IS')
      DO I = 1,N
        CALL Get_U_Exact (U_EXACT, (QFLOAT(I)*H), T)
1402    FORMAT (F36.33,I6,F37.33)

c         WRITE (70,1402)I*H,I,U_EXACT
C           TEMP = ((DREAL(U(I)) - DREAL(U_EXACT))**2 +
C     -             (DIMAG(U(I)) - DIMAG(U_EXACT))**2)
          TEMP = QABS(U(I) - U_EXACT)
          IF (TEMP.GT.NORM) NORM = TEMP
      END DO
C      NORM = (NORM) / QFLOAT(N)
      RETURN
      END
C*********************************************************************C
      SUBROUTINE Get_U_Exact (U,X,T)
C-----------------------------------------------------------------------C
C                                                                       C
C     This subroutine returns the value of u(x,t).                       C
C                                                                       C
C-----------------------------------------------------------------------C
C
      REAL*16 X, U, T, ALPHA
C
      ALPHA = 4.0
      U = T**ALPHA * X * ((1.0) - X)
      RETURN
      END
```

```
C*****************************************************************C
C                  Program  HYP46-SEQ-QUAD.FOR                    C
C*****************************************************************C
C*****************************************************************C
C                                                                 C
C     This program solves the hyperbolic PDE:  Utt - Uxx = f(x,t).C
C     This is a 4th order method which is 6th order for certain   C
C       choices of BETA and TAU.                                  C
C                                                                 C
C*****************************************************************C
C
      INTEGER  BIG_N(20), N, BIG_K(20), I, M, NUM_TIMES
      REAL*16  RIGHT(350), LEFT(350,350), LOWER(350,350),
     -           UPPER(350,350), Z(350), W(350), U(350), U1(350),
     -           U2(350), H, K, BETA(2), ALPHA(2), TAU(2), NORM(20)
C
C
      OPEN (55,FILE='ThesisIn.dat',STATUS='OLD')
      READ (55,*)NUM_TIMES
      DO M = 1,NUM_TIMES
C
      READ (55,*)BIG_N(M),BIG_K(M)
      H = (1.0) / QFLOAT(BIG_N(M)+1)
      K = (1.0) / QFLOAT(BIG_K(M)+1)
C
      CALL Init (BETA, TAU, ALPHA, U, U1, H, K, BIG_N)
102   FORMAT ('   Beta(1) = ',F10.8,'  Beta(2) = ',F10.8)
103   FORMAT ('   Tau(1) = ',F10.8,'  Tau(2) = ',F10.8)
      IF (M.EQ.1) THEN
        WRITE(70,102) BETA(1),BETA(2)
        WRITE(70,103) TAU(1),TAU(2)
      END IF
C
      DO N = 1,BIG_K(M)
        CALL Get_Left_Hand_Side (LEFT,BIG_N(M),K,H,BETA(1))
        CALL Get_Right_Hand_Side (RIGHT,U1,N,BIG_N(M),K,H,TAU,ALPHA)

        CALL Factor (LEFT, LOWER, UPPER, BIG_N(M))
        CALL LSolve (LOWER, Z, RIGHT, BIG_N(M))
        CALL USolve (UPPER, W, Z, BIG_N(M))
C
        CALL Get_Left_Hand_Side (LEFT,BIG_N(M),K,H,BETA(2))
        CALL Factor (LEFT, LOWER, UPPER, BIG_N(M))
        CALL LSolve (LOWER, Z, W, BIG_N(M))
        CALL USolve (UPPER, W, Z, BIG_N(M))
C
        DO I = 1,BIG_N(M)
          U2(I) = W(I) + 2*U1(I) - U(I)
          U(I) = U1(I)
          U1(I) = U2(I)
        END DO
C
      END DO
C
      CALL Error (U2,NORM(M),BIG_N(M),H)
C      WRITE (70,*) ('THE ERROR IS ')
```

```
C       WRITE (70,*) NORM(M)
C
C        Write (6,*)
132    FORMAT (F36.33,I6,F37.33)
c       WRITE (70,*)
c       WRITE (70,*) ('U_APPROXIMATE')
c        DO ROW = 1,BIG_N
c          WRITE (70,132) QFLOAT(ROW)*H,ROW,U2(ROW)
c        END DO
C
       END DO
C
155    FORMAT (I6,I6,F)
156    FORMAT (I6,I6,F,F10.7)
       WRITE(70,155) BIG_N(1)+1,BIG_K(1)+1,NORM(1)
       DO M = 2,NUM_TIMES
         ORDER = ( QLOG( NORM(M-1)/NORM(M) ) / QLOG(QFLOAT(2)) )
         WRITE(70,156) BIG_N(M)+1,BIG_K(M)+1,NORM(M),ORDER
       END DO
C
       STOP
       END
C
C*************************************************************************C
       SUBROUTINE Init (BETA, TAU, ALPHA, U, U1, H, K, BIG_N)
C-------------------------------------------------------------------------C
C                                                                         C
C      This procedure initializes BETA, TAU, ALPHA, U, and U1.            C
C                                                                         C
C-------------------------------------------------------------------------C
C
       REAL*16 BETA(2), TAU(2), ALPHA(2), U(350), U1(350), H, K
       INTEGER BIG_N, I
C
c       BETA(1) = ( QFLOAT(5) + QSQRT(QFLOAT(15)) ) / QFLOAT(60)
c       BETA(2) = ( QFLOAT(5) + QSQRT(QFLOAT(15)) ) / QFLOAT(60)
c       BETA(1) = 0.5
c       BETA(2) = 0.5
c       BETA(2) = qfloat(7)/qfloat(75)
       BETA(1) = qfloat(3)/qfloat(20)
       BETA(2) = (QFLOAT(30)*BETA(1) - QFLOAT(1)) /
      -                (QFLOAT(360)*BETA(1) - QFLOAT(30))
       TAU(2) = 0.1
       TAU(1) = 0.5
c       TAU(1) = qsqrt( ( QFLOAT(2) - QFLOAT(5) * (TAU(2)**2) ) /
c      -           ( QFLOAT(5) * (QFLOAT(1) - QFLOAT(6)*(TAU(2)**2)) ) )
       ALPHA(1) = BETA(1) + BETA(2)
       ALPHA(2) = BETA(1) * BETA(2)
C
c       WRITE (6,*) ('Enter BETA')
c       READ (5,*)BETA
c       WRITE (6,*) ('Enter THETA')
c       READ (5,*)THETA
C
       CALL Clear_Vector (U, BIG_N)
       DO I = 1,BIG_N
```

```
        CALL Get_U_Exact (U1(I), (QFLOAT(I)*H), K)
      END DO
C
      RETURN
      END
C
C*********************************************************************C
      SUBROUTINE Get_Right_Hand_Side (R,U1,N,BIG_N,K,H,TAU,ALPHA)
C---------------------------------------------------------------------C
C                                                                     C
C     This procedure figures up the Right Hand side of the equation   C
C       used in the numerical scheme.  The equation is given in my    C
C       thesis.                                                       C
C     It figures up TEMP1, TEMP2, TEMP3, TEMP4 depending what row     C
C       L is currently at and then                                    C
C        RIGHT = -(TEMP1+TEMP2) + [ (TEMP3+TEMP4)/(TAU(2)^2-TAU(1)^2) ] C
C                                                                     C
C---------------------------------------------------------------------C
C
      REAL*16 R(350), U1(350), K, H, TAU(2), Tn, TEMP1, F1(350), F2(350)
     -        , F_PLUS, F_MINUS, T1, T2, TEMP2, TEMP3, TEMP4, ALPHA(2)
      INTEGER N, BIG_N, L
C
      Tn = QFLOAT(N)*K
C
      T1 = Tn + TAU(1)*K
      T2 = Tn - TAU(1)*K
      DO L = 1,BIG_N
        CALL Get_F (F_PLUS,L,T1,H)
        CALL Get_F (F_MINUS,L,T2,H)
        F1(L) = F_PLUS + F_MINUS
      END DO
C
      T1 = Tn + TAU(2)*K
      T2 = Tn - TAU(2)*K
      DO L = 1,BIG_N
        CALL Get_F (F_PLUS,L,T1,H)
        CALL Get_F (F_MINUS,L,T2,H)
        F2(L) = F_PLUS + F_MINUS
      END DO
C
      L = 1
        TEMP1 = ((K/H)**2)*(2*U1(L) - U1(L+1))
        TEMP3 = (TAU(2)**2)*( .5*(K**2)*F1(L) + ( .5*ALPHA(1) -
     -                        QFLOAT(1)/QFLOAT(24) )*(K**4)*(H**(-2))
     -                        *( 2*F1(L) - F1(L+1) ) )
     -        - ( (QFLOAT(1)/QFLOAT(12))*(K**2)*F1(L) +
     -                        ALPHA(2)*(K**4)*(H**(-2))
     -                        *( 2*F1(L) - F1(L+1) ) )
        TEMP4 = ( (QFLOAT(1)/QFLOAT(12))*(K**2)*F2(L) +
     -                        ALPHA(2)*(K**4)*(H**(-2))
     -                        *( 2*F2(L) - F2(L+1) ) )
     -        - (TAU(1)**2)*( .5*(K**2)*F2(L) + ( .5*ALPHA(1) -
     -                        QFLOAT(1)/QFLOAT(24) )*(K**4)*(H**(-2))
     -                        *( 2*F2(L) - F2(L+1) ) )
        IF (BIG_N.EQ.2) THEN
```

```
          TEMP2 = (ALPHA(1) - QFLOAT(1)/QFLOAT(12))*((K/H)**4)*
     -            (5*U1(L) - 4*U1(L+1))
          ELSE
          TEMP2 = (ALPHA(1) - QFLOAT(1)/QFLOAT(12))*((K/H)**4)*
     -            (5*U1(L) - 4*U1(L+1) + U1(L+2))
          END IF
          R(L) = -(TEMP1 + TEMP2) + (TEMP3 + TEMP4) /
     -                          (TAU(2)**2 - TAU(1)**2)
C
          L = BIG_N
          TEMP1 = ((K/H)**2)*(-U1(L-1) + 2*U1(L))
          TEMP3 = (TAU(2)**2)*( .5*(K**2)*F1(L) + ( .5*ALPHA(1) -
     -                          QFLOAT(1)/QFLOAT(24) )*(K**4)*(H**(-2))
     -                          *( -F1(L-1) + 2*F1(L) ) )
     -            - ( (QFLOAT(1)/QFLOAT(12))*(K**2)*F1(L) +
     -                          ALPHA(2)*(K**4)*(H**(-2))
     -                          *( -F1(L-1) + 2*F1(L) ) )
          TEMP4 = ( (QFLOAT(1)/QFLOAT(12))*(K**2)*F2(L) +
     -                          ALPHA(2)*(K**4)*(H**(-2))
     -                          *( -F2(L-1) + 2*F2(L) ) )
     -            - (TAU(1)**2)*( .5*(K**2)*F2(L) + ( .5*ALPHA(1) -
     -                          QFLOAT(1)/QFLOAT(24) )*(K**4)*(H**(-2))
     -                          *( -F2(L-1) + 2*F2(L) ) )
          IF (BIG_N.EQ.2) THEN
          TEMP2 = (ALPHA(1) - QFLOAT(1)/QFLOAT(12))*((K/H)**4)*
     -            (-4*U1(L-1) + 5*U1(L))
          ELSE
          TEMP2 = (ALPHA(1) - QFLOAT(1)/QFLOAT(12))*((K/H)**4)*
     -            (1*U1(L-2) - 4*U1(L-1) + 5*U1(L))
          END IF
          R(L) = -(TEMP1 + TEMP2) + (TEMP3 + TEMP4) /
     -                           (TAU(2)**2 - TAU(1)**2)

C
          DO L = 2,BIG_N-1
          TEMP1 = ((K/H)**2)*(-U1(L-1) + 2*U1(L) - U1(L+1))
          TEMP3 = (TAU(2)**2)*( .5*(K**2)*F1(L) + ( .5*ALPHA(1) -
     -                          QFLOAT(1)/QFLOAT(24) )*(K**4)*(H**(-2))
     -                          *( -F1(L-1) + 2*F1(L) - F1(L+1) ) )
     -            - ( (QFLOAT(1)/QFLOAT(12))*(K**2)*F1(L) +
     -                          ALPHA(2)*(K**4)*(H**(-2))
     -                          *( -F1(L-1) + 2*F1(L) - F1(L+1) ) )
          TEMP4 = ( (QFLOAT(1)/QFLOAT(12))*(K**2)*F2(L) +
     -                          ALPHA(2)*(K**4)*(H**(-2))
     -                          *( -F2(L-1) + 2*F2(L) - F2(L+1) ) )
     -            - (TAU(1)**2)*( .5*(K**2)*F2(L) + ( .5*ALPHA(1) -
     -                          QFLOAT(1)/QFLOAT(24) )*(K**4)*(H**(-2))
     -                          *( -F2(L-1) + 2*F2(L) - F2(L+1) ) )
          IF (L.EQ.2) THEN
            IF (BIG_N.EQ.3) THEN
              TEMP2 = (ALPHA(1) - QFLOAT(1)/QFLOAT(12))*((K/H)**4)*
     -                (-4*U1(L-1) + 6*U1(L) - 4*U1(L+1))
            ELSE
              TEMP2 = (ALPHA(1) - QFLOAT(1)/QFLOAT(12))*((K/H)**4)*
     -                (-4*U1(L-1) + 6*U1(L) - 4*U1(L+1) + U1(L+2))
            END IF
```

```
         ELSE IF (L.EQ.BIG_N-1) THEN
           TEMP2 = (ALPHA(1) - QFLOAT(1)/QFLOAT(12))*((K/H)**4)*
     -             (U1(L-2) - 4*U1(L-1) + 6*U1(L) - 4*U1(L+1))
         ELSE
           TEMP2 = (ALPHA(1) - QFLOAT(1)/QFLOAT(12))*((K/H)**4)*
     -             (U1(L-2) - 4*U1(L-1) + 6*U1(L) - 4*U1(L+1) + U1(L+2))
         END IF
         R(L) = -(TEMP1 + TEMP2) + (TEMP3 + TEMP4) /
     -                           (TAU(2)**2 - TAU(1)**2)
       END DO
C
       RETURN
       END
C
C*************************************************************************C
       SUBROUTINE Get_F (F,I,T,H)
C-------------------------------------------------------------------------C
C                                                                         C
C      This procedure gets the value of F(X,T) where F is the value to    C
C        the right of the PDE, X = IxH, and T = T.                        C
C                                                                         C
C-------------------------------------------------------------------------C

C
       REAL*16 H, T, F, ALPHA, X
       INTEGER I
C
       ALPHA = (4.0)
       X = QFLOAT(I) * H
       F = (T**(ALPHA-(2.0))) * (ALPHA*(ALPHA-1.0)*X*((1.0)-X)+
     -    (2.0)*(T**2))
C
       RETURN
       END
C
C*************************************************************************C
       SUBROUTINE Get_Left_Hand_Side (L,BIG_N,K,H,BETA)
C-------------------------------------------------------------------------C
C                                                                         C
C      This procedure figures up the left side of the numerical scheme    C
C        which happens to be (I-BETA K^2 Lh).  Where Lh is your           C
C        traditional finite difference matrix -DIAGONAL(-1 2 -1).         C
C                                                                         C
C-------------------------------------------------------------------------C
C
       REAL*16 L(350,350), H, K, BETA
       INTEGER  BIG_N, ROW
C
       CALL Clear_Matrix (L, BIG_N)
       IF (BIG_N.EQ.1) THEN
         ROW = 1
         L(ROW,ROW+1) = - BETA*(K**2)*(H**(-2.0))
         L(ROW,ROW) = (1.0) + BETA*(K**2)*2.0*(H**(-2.0))
       ELSE
         ROW = 1
         L(ROW,ROW+1) = - BETA*(K**2)*(H**(-2.0))
```

```
        L(ROW,ROW) = (1.0) + BETA*(K**2)*2.0*(H**(-2.0))
        ROW = BIG_N
        L(ROW,ROW) = (1.0) + BETA*(K**2)*2.0*(H**(-2.0))
        L(ROW,ROW-1) = - BETA*(K**2)*(H**(-2.0))
        DO ROW = 2,BIG_N-1
          L(ROW,ROW+1) = - BETA*(K**2)*(H**(-2.0))
          L(ROW,ROW) = (1.0) + BETA*(K**2)*2.0*(H**(-2.0))
          L(ROW,ROW-1) = - BETA*(K**2)*(H**(-2.0))
        END DO
      END IF
C
      RETURN
      END
C
C*************************************************************************C
      SUBROUTINE Clear_Matrix (A, N)
C-------------------------------------------------------------------------C
C                                                                         C
C     This procedure sets all entries in matrix A to (0.0).               C
C                                                                         C
C-------------------------------------------------------------------------C
C
      REAL*16 A(350,350)
      INTEGER ROW, COL, N
C
      DO ROW = 1,N
        DO COL = 1,N
          A(ROW,COL) = (0.0)
        END DO
      END DO
      RETURN
      END
C
C*************************************************************************C
      SUBROUTINE CLEAR_VECTOR (A, N)
C-------------------------------------------------------------------------C
C                                                                         C
C     This procedure sets all entries in vector A to (0.0).               C
C                                                                         C
C-------------------------------------------------------------------------C
C
      REAL*16 A(350)
      INTEGER ROW, N
C
      DO ROW = 1,N
          A(ROW) = (0.0)
      END DO
      RETURN
      END
C
C*************************************************************************C
      SUBROUTINE FACTOR(A, L, U, N)
C-------------------------------------------------------------------------C
C                                                                         C
C     This procedure factors A into L and U where L is a lower            C
```

```
C     triangular matrix and U is an upper triangular matrix.            C
C                                                                       C
C-----------------------------------------------------------------------C
C
      REAL*16 TEMP, A(350,350), L(350,350), U(350,350)
      INTEGER I, J, K, N
C
      DO I = 1,N
        L(I,I) = (1.0)
      END DO
C
      U(1,1) = A(1,1)
      IF (U(1,1).EQ.(0.0)) THEN
1100     FORMAT(' 1  FACTORIZATION IMPOSSIBLE!')
1101     FORMAT(' 2  FACTORIZATION IMPOSSIBLE!')
        WRITE (6,1100)
        STOP
      END IF
C
      DO I = 2,N
        U(1,I) = A(1,I) / L(1,1)
        L(I,1) = A(I,1) / U(1,1)
      END DO
C
      DO I = 2,N-1
        TEMP = (0.0)
        DO K = 1,I-1
          TEMP = TEMP + (L(I,K)*U(K,I))
        END DO
        U(I,I) = A(I,I) - TEMP
        IF (U(I,I).EQ.(0.0)) THEN
          WRITE (6,1101)
          STOP
        END IF
C
        DO J = I+1,N
          TEMP = (0.0)
          DO K = 1,I-1
            TEMP = TEMP + (L(I,K) * U(K,J))
          END DO
          U(I,J) = (A(I,J) - TEMP) / L(I,I)
C
          TEMP = (0.0)
          DO K = 1,I-1
            TEMP = TEMP + (L(J,K) * U(K,I))
          END DO
          L(J,I) = (A(J,I) - TEMP) / U(I,I)
        END DO
      END DO
C
      TEMP = (0.0)
      dO K = 1,N-1
        TEMP = TEMP + (L(N,K) * U(K,N))
      END DO
      U(N,N) = A(N,N) - TEMP
      RETURN
```

```
      END
C
C*************************************************************************C
      SUBROUTINE TRANSPOSE (A,TRANS,N)
C-------------------------------------------------------------------------C
C                                                                         C
C     This procedure figures the transpose of matrix A and puts the       C
C     answer in TRANS.                                                    C
C                                                                         C
C-------------------------------------------------------------------------C
C
      REAL*16 A(350,350), TRANS(350,350)
      INTEGER N, ROW, COL
C
      DO ROW = 1,N
        DO COL = 1,N
          TRANS(COL,ROW) = A(ROW,COL)
        END DO
      END DO
      RETURN
      END
C
C*************************************************************************C
      SUBROUTINE USOLVE(U, W, X, N)
C-------------------------------------------------------------------------C
C                                                                         C
C     This procedure takes an upper triangular matrix, U, and solves      C
C     the equation UW=X for W.                                            C
C                                                                         C
C-------------------------------------------------------------------------C
C
      REAL*16 U(350,350), W(350), X(350), TEMP
      INTEGER I, J, K, N
C
      W(N) = X(N) / U(N,N)
C
      DO I = N - 1,1,-1
        DO J = 1,N
          TEMP = (0.0)
          DO K = I + 1,N
            TEMP = TEMP + (U(I,K)*W(K))
          END DO
          W(I) = (X(I) - TEMP) / U(I,I)
        END DO
      END DO
      RETURN
      END
C
C*************************************************************************C
      SUBROUTINE LSOLVE(L, M, W, N)
C-------------------------------------------------------------------------C
C                                                                         C
C     This procedure takes a lower triangular matrix, L, and solves the   C
C     equation LM=W for M.                                                C
C                                                                         C
C-------------------------------------------------------------------------C
```

```
C
      REAL*16 L(350,350), M(350), W(350), TEMP
      INTEGER I, J, K, N
C
      DO I = 1,N
        M(1) = W(1)
      END DO
C
      DO I = 2,N
        DO J = 1,N
          TEMP = (0.0)
          DO K = 1,I - 1
            TEMP = TEMP + (L(I,K)*M(K))
          END DO
          M(I) = (W(I) - TEMP)
        END DO
      END DO
      RETURN
      END
C
C*********************************************************************C
      SUBROUTINE Error (U,NORM,N,H)
C-------------------------------------------------------------------C
C                                                                   C
C     This procedure figures the error between U and U_EXACT and    C
C       puts the answer in NORM.                                    C
C                                                                   C
C-------------------------------------------------------------------C
C
      REAL*16 U(350), U_EXACT, NORM, TEMP, H, T
      INTEGER N, I
C
      NORM = (0.0)
      T = (1.0)
      NORM = 0.0
c      WRITE (70,*) ('U_EXACT IS')
      DO I = 1,N
        CALL Get_U_Exact (U_EXACT, (QFLOAT(I)*H), T)
1402    FORMAT (F36.33,I6,F37.33)

c         WRITE (70,1402)I*H,I,U_EXACT
C           TEMP = ((DREAL(U(I)) - DREAL(U_EXACT))**2 +
C     -             (DIMAG(U(I)) - DIMAG(U_EXACT))**2)
          TEMP = QABS(U(I) - U_EXACT)
          IF (TEMP.GT.NORM) NORM = TEMP
      END DO
C      NORM = (NORM) / QFLOAT(N)
      RETURN
      END
C*********************************************************************C
      SUBROUTINE Get_U_Exact (U,X,T)
C-------------------------------------------------------------------C
C                                                                   C
C     This subroutine returns the value of u(x,t).                 C
C                                                                   C
C-------------------------------------------------------------------C
```

```
C
      REAL*16 X, U, T, ALPHA
C
      ALPHA = 4.0
      U = T**ALPHA * X * ((1.0) - X)
      RETURN
      END
```

```
C*****************************************************************C
C                  Program  HYP46-PF-QUAD.FOR                     C
C*****************************************************************C
C*****************************************************************C
C                                                                 C
C     This program solves the hyperbolic PDE:  Utt - Uxx = f(x,t).C
C     This is a 4th order method which is 6th order for certain   C
C       choices of BETA and TAU.                                  C
C     This program uses the partial fractions approach and is the C
C       first step in parallelizing this method.                  C
C                                                                 C
C*****************************************************************C
C
      INTEGER  BIG_N(20), N, BIG_K(20), I, M, NUM_TIMES
      REAL*16  RIGHT(350), LEFT(350,350), LOWER(350,350),
     -           UPPER(350,350), Z(350), W(350), U(350), U1(350),
     -           U2(350), H, K, BETA(2), ALPHA(2), TAU(2), NORM(20),
     -           A0, A(2), C(2), D(2), Y1(350), Y2(350)
C
C
      OPEN (55,FILE='ThesisIn2.dat',STATUS='OLD')
      READ (55,*)NUM_TIMES
      DO M = 1,NUM_TIMES
C
      READ (55,*)BIG_N(M),BIG_K(M)
      H = (1.0) / QFLOAT(BIG_N(M)+1)
      K = (1.0) / QFLOAT(BIG_K(M)+1)
C
      CALL Init (BETA, TAU, ALPHA, U, U1, H, K, BIG_N(M), A0, A,
     -           C, D)
102   FORMAT ('   Beta(1) = ',F10.8,'  Beta(2) = ',F10.8)
103   FORMAT ('   Tau(1) = ',F10.8,'  Tau(2) = ',F10.8)
      IF (M.EQ.1) THEN
        WRITE(6,102) BETA(1),BETA(2)
        WRITE(6,103) TAU(1),TAU(2)
      END IF
C
      DO N = 1,BIG_K(M)
        CALL Get_Left_Hand_Side (LEFT,BIG_N(M),K,H,BETA(1))
        CALL Get_Right_Hand_Side (RIGHT,U1,N,BIG_N(M),K,H,TAU,A(1),
     -                            C(1),D(1))

        CALL Factor (LEFT, LOWER, UPPER, BIG_N(M))
        CALL LSolve (LOWER, Z, RIGHT, BIG_N(M))
        CALL USolve (UPPER, Y1, Z, BIG_N(M))
C
        CALL Get_Left_Hand_Side (LEFT,BIG_N(M),K,H,BETA(2))
        CALL Get_Right_Hand_Side (RIGHT,U1,N,BIG_N(M),K,H,TAU,A(2),
     -                            C(2),D(2))
        CALL Factor (LEFT, LOWER, UPPER, BIG_N(M))
        CALL LSolve (LOWER, Z, RIGHT, BIG_N(M))
        CALL USolve (UPPER, Y2, Z, BIG_N(M))
C
        DO I = 1,BIG_N(M)
          W(I) = Y1(I) + Y2(I)
          U2(I) = W(I) + 2*A0*U1(I) - U(I)
```

```
          U(I) = U1(I)
          U1(I) = U2(I)
        END DO
C
      END DO
C
      CALL Error (U2,NORM(M),BIG_N(M),H)
C      WRITE (70,*) ('THE ERROR IS ')
C      WRITE (70,*) NORM(M)
C
C       Write (6,*)
132   FORMAT (F36.33,I6,F37.33)
c      WRITE (70,*)
c      WRITE (70,*) ('U_APPROXIMATE')
c       DO ROW = 1,BIG_N
c         WRITE (70,132) QFLOAT(ROW)*H,ROW,U2(ROW)
c       END DO
C
      END DO
C
155   FORMAT (I6,I6,F)
156   FORMAT (I6,I6,F,F10.7)
      WRITE(6,155) BIG_N(1)+1,BIG_K(1)+1,NORM(1)
      DO M = 2,NUM_TIMES
        ORDER = ( QLOG( NORM(M-1)/NORM(M) ) / QLOG(QFLOAT(2)) )
        WRITE(6,156) BIG_N(M)+1,BIG_K(M)+1,NORM(M),ORDER
      END DO
C
      STOP
      END
C
C*************************************************************************C
      SUBROUTINE Init (BETA, TAU, ALPHA, U, U1, H, K, BIG_N, A0, A, C,D)
C-------------------------------------------------------------------------C
C                                                                         C
C     This procedure initializes BETA, TAU, ALPHA, A0, A, C, D, U,        C
C       and U1.                                                           C
C                                                                         C
C-------------------------------------------------------------------------C
C
      REAL*16 BETA(2), TAU(2), ALPHA(2), U(350), U1(350), H, K,
     -          A0, A(2), B, C(2), D(2)
      INTEGER BIG_N, I
C
c      BETA(1) = ( QFLOAT(5) + QSQRT(QFLOAT(15)) ) / QFLOAT(60) +.1
      BETA(1) = QFLOAT(3) / QFLOAT(20)
      BETA(2) = (QFLOAT(30)*BETA(1) - QFLOAT(1)) /
     -               (QFLOAT(360)*BETA(1) - QFLOAT(30))
      TAU(2) = 0.99
      TAU(1) = qsqrt( ( QFLOAT(2) - QFLOAT(5) * (TAU(2)**2) ) /
     -           ( QFLOAT(5) * (QFLOAT(1) - QFLOAT(6)*(TAU(2)**2)) ) )
      ALPHA(1) = BETA(1) + BETA(2)
      ALPHA(2) = BETA(1) * BETA(2)
C
      A0 = (ALPHA(2) - (ALPHA(1)/QFLOAT(2)) + (QFLOAT(1)/QFLOAT(24)))
     -      / ALPHA(2)
```

```
      A(1) = (QFLOAT(1) - QFLOAT(12) * BETA(2)) /
     -       (QFLOAT(24)*BETA(1) * (BETA(1)-BETA(2)) )
      A(2) = (QFLOAT(1) - QFLOAT(12) * BETA(1)) /
     -       (QFLOAT(24)*BETA(2) * (BETA(2)-BETA(1)) )
      B = (K**2) / ( (TAU(2)**2 - TAU(1)**2) * (BETA(1) - BETA(2)) )
      C(1) = B * ( BETA(1)/QFLOAT(12) - ALPHA(2) + TAU(1)**2*
     -             (ALPHA(1)/QFLOAT(2) - QFLOAT(1)/QFLOAT(24)
     -                         - BETA(1)/QFLOAT(2)) )
      C(2) = B * ( ALPHA(2) - BETA(2)/QFLOAT(12) - TAU(1)**2*
     -             (ALPHA(1)/QFLOAT(2) - QFLOAT(1)/QFLOAT(24)
     -                         - BETA(2)/QFLOAT(2)) )
      D(1) = B * ( -BETA(1)/QFLOAT(12) + ALPHA(2) + TAU(2)**2*
     -             (-ALPHA(1)/QFLOAT(2) + QFLOAT(1)/QFLOAT(24)
     -                         + BETA(1)/QFLOAT(2)) )
      D(2) = B * ( -ALPHA(2) + BETA(2)/QFLOAT(12) + TAU(2)**2*
     -             (ALPHA(1)/QFLOAT(2) - QFLOAT(1)/QFLOAT(24)
     -                         - BETA(2)/QFLOAT(2)) )

C
      CALL Clear_Vector (U, BIG_N)
      DO I = 1,BIG_N
        CALL Get_U_Exact (U1(I), (QFLOAT(I)*H), K)
      END DO
C
      RETURN
      END
C
C*************************************************************************C
      SUBROUTINE Get_Right_Hand_Side (R,U1,N,BIG_N,K,H,TAU,A,C,D)
C-------------------------------------------------------------------------C
C                                                                         C
C     This procedure figures up the Right Hand side of the equation       C
C       used in the numerical scheme.  The equation is given in my        C
C       thesis.                                                           C
C     It figures up RIGHT depending what row L is currently at and then C
C        RIGHT = 2A U1 + K^2 D Fh(TAU(1),K) + K^2 C Fh(TAU(2),K)          C
C                                                                         C
C-------------------------------------------------------------------------C
C
      REAL*16 R(350), U1(350), K, H, TAU(2), Tn, F1(350), F2(350)
     -        , F_PLUS, F_MINUS, T1, T2, A, C, D
      INTEGER N, BIG_N, L
C
      Tn = QFLOAT(N)*K
C
      T1 = Tn + TAU(1)*K
      T2 = Tn - TAU(1)*K
      DO L = 1,BIG_N
        CALL Get_F (F_PLUS,L,T1,H)
        CALL Get_F (F_MINUS,L,T2,H)
        F1(L) = F_PLUS + F_MINUS
      END DO
C
      T1 = Tn + TAU(2)*K
      T2 = Tn - TAU(2)*K
      DO L = 1,BIG_N
```

```
        CALL Get_F (F_PLUS,L,T1,H)
        CALL Get_F (F_MINUS,L,T2,H)
        F2(L) = F_PLUS + F_MINUS
      END DO
C
      DO L = 1,BIG_N
        R(L) = 2*A*U1(L) + D*F1(L) + C*F2(L)
      END DO
C
      RETURN
      END
C
C*********************************************************************C
      SUBROUTINE Get_F (F,I,T,H)
C---------------------------------------------------------------------C
C                                                                     C
C     This procedure gets the value of F(X,T) where F is the value to C
C       the right of the PDE, X = IxH, and T = T.                     C
C                                                                     C
C---------------------------------------------------------------------C

C
      REAL*16 H, T, F, ALPHA, X
      INTEGER I
C
      ALPHA = (4.0)
      X = QFLOAT(I) * H
      F = (T**(ALPHA-(2.0))) * (ALPHA*(ALPHA-1.0)*X*((1.0)-X)+
     -    (2.0)*(T**2))
C
      RETURN
      END
C
C*********************************************************************C
      SUBROUTINE Get_Left_Hand_Side (L,BIG_N,K,H,BETA)
C---------------------------------------------------------------------C
C                                                                     C
C     This procedure figures up the left side of the numerical scheme C
C       which happens to be (I-BETA K^2 Lh).  Where Lh is your        C
C       traditional finite difference matrix -DIAGONAL(-1 2 -1).      C
C                                                                     C
C---------------------------------------------------------------------C
C
      REAL*16 L(350,350), H, K, BETA
      INTEGER  BIG_N, ROW
C
      CALL Clear_Matrix (L, BIG_N)
      IF (BIG_N.EQ.1) THEN
        ROW = 1
        L(ROW,ROW+1) = - BETA*(K**2)*(H**(-2.0))
        L(ROW,ROW) = (1.0) + BETA*(K**2)*2.0*(H**(-2.0))
      ELSE
        ROW = 1
        L(ROW,ROW+1) = - BETA*(K**2)*(H**(-2.0))
        L(ROW,ROW) = (1.0) + BETA*(K**2)*2.0*(H**(-2.0))
        ROW = BIG_N
```

```
          L(ROW,ROW) = (1.0) + BETA*(K**2)*2.0*(H**(-2.0))
          L(ROW,ROW-1) = - BETA*(K**2)*(H**(-2.0))
          DO ROW = 2,BIG_N-1
            L(ROW,ROW+1) = - BETA*(K**2)*(H**(-2.0))
            L(ROW,ROW) = (1.0) + BETA*(K**2)*2.0*(H**(-2.0))
            L(ROW,ROW-1) = - BETA*(K**2)*(H**(-2.0))
          END DO
        END IF
C
        RETURN
        END
C
C*************************************************************************C
        SUBROUTINE Clear_Matrix (A, N)
C-------------------------------------------------------------------------C
C                                                                         C
C     This procedure sets all entries in matrix A to (0.0).               C
C                                                                         C
C-------------------------------------------------------------------------C
C
        REAL*16 A(350,350)
        INTEGER ROW, COL, N
C
        DO ROW = 1,N
          DO COL = 1,N
            A(ROW,COL) = (0.0)
          END DO
        END DO
        RETURN
        END
C
C*************************************************************************C
        SUBROUTINE CLEAR_VECTOR (A, N)
C-------------------------------------------------------------------------C
C                                                                         C
C     This procedure sets all entries in vector A to (0.0).               C
C                                                                         C
C-------------------------------------------------------------------------C
C
        REAL*16 A(350)
        INTEGER ROW, N
C
        DO ROW = 1,N
            A(ROW) = (0.0)
        END DO
        RETURN
        END
C
C*************************************************************************C
        SUBROUTINE FACTOR(A, L, U, N)
C-------------------------------------------------------------------------C
C                                                                         C
C     This procedure factors A into L and U where L is a lower            C
C     triangular matrix and U is an upper triangular matrix.              C
C                                                                         C
```

```
C-----------------------------------------------------------------------C
C
      REAL*16 TEMP, A(350,350), L(350,350), U(350,350)
      INTEGER I, J, K, N
C
      DO I = 1,N
        L(I,I) = (1.0)
      END DO
C
      U(1,1) = A(1,1)
      IF (U(1,1).EQ.(0.0)) THEN
1100     FORMAT(' 1  FACTORIZATION IMPOSSIBLE!')
1101     FORMAT(' 2  FACTORIZATION IMPOSSIBLE!')
        WRITE (6,1100)
        STOP
      END IF
C
      DO I = 2,N
        U(1,I) = A(1,I) / L(1,1)
        L(I,1) = A(I,1) / U(1,1)
      END DO
C
      DO I = 2,N-1
        TEMP = (0.0)
        DO K = 1,I-1
          TEMP = TEMP + (L(I,K)*U(K,I))
        END DO
        U(I,I) = A(I,I) - TEMP
        IF (U(I,I).EQ.(0.0)) THEN
          WRITE (6,1101)
          STOP
        END IF
C
        DO J = I+1,N
          TEMP = (0.0)
          DO K = 1,I-1
            TEMP = TEMP + (L(I,K) * U(K,J))
          END DO
          U(I,J) = (A(I,J) - TEMP) / L(I,I)
C
          TEMP = (0.0)
          DO K = 1,I-1
            TEMP = TEMP + (L(J,K) * U(K,I))
          END DO
          L(J,I) = (A(J,I) - TEMP) / U(I,I)
        END DO
      END DO
C
      TEMP = (0.0)
      dO K = 1,N-1
        TEMP = TEMP + (L(N,K) * U(K,N))
      END DO
      U(N,N) = A(N,N) - TEMP
      RETURN
      END
C
```

```
C*********************************************************************C
      SUBROUTINE TRANSPOSE (A,TRANS,N)
C---------------------------------------------------------------------C
C                                                                     C
C     This procedure figures the transpose of matrix A and puts the   C
C     answer in TRANS.                                                C
C                                                                     C
C---------------------------------------------------------------------C
C
      REAL*16 A(350,350), TRANS(350,350)
      INTEGER N, ROW, COL
C
      DO ROW = 1,N
        DO COL = 1,N
          TRANS(COL,ROW) = A(ROW,COL)
        END DO
      END DO
      RETURN
      END
C
C*********************************************************************C
      SUBROUTINE USOLVE(U, W, X, N)
C---------------------------------------------------------------------C
C                                                                     C
C     This procedure takes an upper triangular matrix, U, and solves  C
C     the equation UW=X for W.                                        C
C                                                                     C
C---------------------------------------------------------------------C
C
      REAL*16 U(350,350), W(350), X(350), TEMP
      INTEGER I, J, K, N
C
      W(N) = X(N) / U(N,N)
C
      DO I = N - 1,1,-1
        DO J = 1,N
          TEMP = (0.0)
          DO K = I + 1,N
            TEMP = TEMP + (U(I,K)*W(K))
          END DO
          W(I) = (X(I) - TEMP) / U(I,I)
        END DO
      END DO
      RETURN
      END
C
C*********************************************************************C
      SUBROUTINE LSOLVE(L, M, W, N)
C---------------------------------------------------------------------C
C                                                                     C
C     This procedure takes a lower triangular matrix, L, and solves the C
C     equation LM=W for M.                                            C
C                                                                     C
C---------------------------------------------------------------------C
C
      REAL*16 L(350,350), M(350), W(350), TEMP
```

```
      INTEGER I, J, K, N
C
      DO I = 1,N
        M(1) = W(1)
      END DO
C
      DO I = 2,N
        DO J = 1,N
          TEMP = (0.0)
          DO K = 1,I - 1
            TEMP = TEMP + (L(I,K)*M(K))
          END DO
          M(I) = (W(I) - TEMP)
        END DO
      END DO
      RETURN
      END
C
C*************************************************************************C
      SUBROUTINE Error (U,NORM,N,H)
C-------------------------------------------------------------------------C
C                                                                         C
C     This procedure figures the error between U and U_EXACT and          C
C       puts the answer in NORM.                                          C
C                                                                         C
C-------------------------------------------------------------------------C
C
      REAL*16 U(350), U_EXACT, NORM, TEMP, H, T
      INTEGER N, I
C
      NORM = (0.0)
      T = (1.0)
      NORM = 0.0
c      WRITE (70,*) ('U_EXACT IS')
      DO I = 1,N
        CALL Get_U_Exact (U_EXACT, (QFLOAT(I)*H), T)
1402    FORMAT (F36.33,I6,F37.33)

c         WRITE (70,1402)I*H,I,U_EXACT
C           TEMP = ((DREAL(U(I)) - DREAL(U_EXACT))**2 +
C     -             (DIMAG(U(I)) - DIMAG(U_EXACT))**2)
          TEMP = QABS(U(I) - U_EXACT)
          IF (TEMP.GT.NORM) NORM = TEMP
      END DO
C      NORM = (NORM) / QFLOAT(N)
      RETURN
      END
C*************************************************************************C
      SUBROUTINE Get_U_Exact (U,X,T)
C-------------------------------------------------------------------------C
C                                                                         C
C     This subroutine returns the value of u(x,t).                        C
C                                                                         C
C-------------------------------------------------------------------------C
C
      REAL*16 X, U, T, ALPHA
```

```
C
      ALPHA = 4.0
      U = T**ALPHA * X * ((1.0) - X)
      RETURN
      END
```

```
      PROGRAM hypmaster
C*********************************************************************C
C                                                                     C
C     This program solves the hyperbolic PDE:  Utt - Uxx = f(x,t).   C
C     This is a 4th order method which is 6th order for certain       C
C       choices of BETA and TAU.                                      C
C     This program uses the partial fractions approach and is the     C
C       parallelized method using PVM.  This program also requires    C
C       hyp-worker.                                                   C
C                                                                     C
C*********************************************************************C
      INCLUDE 'pvm3/include/fpvm3.h'
C
      INTEGER  J, BIG_N(20), N, BIG_K(20), I, M, NUM_TIMES, MAXBIG_N
      INTEGER NPROCS
      PARAMETER (NPROCS = 2, MAXBIG_N = 350)
      INTEGER FLAG, mytid, task_ids(NPROCS), numt, info, msgtype
      REAL*8  W(MAXBIG_N), U(MAXBIG_N), U1(MAXBIG_N),
     -        U2(MAXBIG_N), H, K, BETA(2), ALPHA(2), TAU(2), NORM(20),
     -        A0, A(2), C(2), D(2), Y1(MAXBIG_N), Y2(MAXBIG_N)
C
C
      OPEN (55,FILE='ThesisIn.dat',STATUS='OLD')
      READ (55,*)NUM_TIMES
      DO M = 1,NUM_TIMES
C
        READ (55,*)BIG_N(M),BIG_K(M)
        H = (1.0) / DFLOAT(BIG_N(M)+1)
        K = (1.0) / DFLOAT(BIG_K(M)+1)
C
        CALL Init (BETA, TAU, ALPHA, U, U1, H, K, BIG_N(M), A0, A,
     -             C, D)
102     FORMAT ('   Beta(1) = ',F10.8,'  Beta(2) = ',F10.8)
103     FORMAT ('   Tau(1) = ',F10.8,'  Tau(2) = ',F10.8)
        IF (M.EQ.1) THEN
          WRITE(75,102) BETA(1),BETA(2)
          WRITE(75,103) TAU(1),TAU(2)
        END IF
C
        DO N = 1,BIG_K(M)
C     * Enroll in PVM *
        CALL pvmfmytid(mytid)
C
        FLAG = PVMDEFAULT
C
C     * Spawn worker tasks *
        CALL pvmfspawn('hyp-worker', FLAG, '', NPROCS, task_ids, numt)
        msgtype = 1

        DO J = 1,NPROCS
          CALL pvmfinitsend(PVMDEFAULT, info)
          CALL pvmfpack(3, BIG_N(M), 1, 1, info)
          CALL pvmfpack(6, U1, MAXBIG_N, 1, info)
          CALL pvmfpack(6, K, 1, 1, info)
          CALL pvmfpack(6, H, 1, 1, info)
          CALL pvmfpack(6, A(J), 1, 1, info)
```

```
            CALL pvmfpack(6, H, 1, 1, info)
            CALL pvmfpack(6, A(J), 1, 1, info)
            CALL pvmfpack(6, C(J), 1, 1, info)
            CALL pvmfpack(6, D(J), 1, 1, info)
            CALL pvmfpack(6, TAU, 2, 1, info)
            CALL pvmfpack(6, BETA(J), 1, 1, info)
            CALL pvmfpack(3, N, 1, 1, info)
            CALL pvmfsend (task_ids(j), msgtype, info)
          END DO
C
C         * Get results *
          msgtype = 7
            CALL pvmfrecv(task_ids(1), msgtype, info)
            CALL pvmfunpack(6, Y1, MAXBIG_N, 1, info)
C
            CALL pvmfrecv(task_ids(2), msgtype, info)
            CALL pvmfunpack(6, Y2, MAXBIG_N, 1, info)
C
C         * Put together U2 *
            DO I = 1,BIG_N(M)
              W(I) = Y1(I) + Y2(I)
              U2(I) = W(I) + 2*A0*U1(I) - U(I)
              U(I) = U1(I)
              U1(I) = U2(I)
            END DO
C
          END DO
C
        CALL Error (U2,NORM(M),BIG_N(M),H)
C        WRITE (75,*) ('THE ERROR IS ')
C        WRITE (75,*) NORM(M)
C
C         Write (6,*)
132     FORMAT (F36.33,I6,F37.33)
c        WRITE (75,*)
c        WRITE (75,*) ('U_APPROXIMATE')
c         DO ROW = 1,BIG_N
c           WRITE (75,132) DFLOAT(ROW)*H,ROW,U2(ROW)
c         END DO
C
        END DO
C
155     FORMAT (I6,I6,F)
156     FORMAT (I6,I6,F,F10.7)
        WRITE(75,155) BIG_N(1)+1,BIG_K(1)+1,NORM(1)
        DO M = 2,NUM_TIMES
          ORDER = ( DLOG( NORM(M-1)/NORM(M) ) / DLOG(DFLOAT(2)) )
          WRITE(75,156) BIG_N(M)+1,BIG_K(M)+1,NORM(M),ORDER
        END DO
C
        CALL pvmfexit(info)
        STOP
        END

C
C*******************************************************************************C
```

```
C                                                                       C
C     This procedure initializes BETA, TAU, ALPHA, A0, A, C, D, U,      C
C       and U1.                                                         C
C                                                                       C
C-----------------------------------------------------------------------C
C
      INTEGER  MAXBIG_N
      PARAMETER (MAXBIG_N = 350)
      REAL*8 BETA(2), TAU(2), ALPHA(2), U(MAXBIG_N), U1(MAXBIG_N), H,
     -        K, A0, A(2), B, C(2), D(2)
      INTEGER BIG_N, I
C
c      BETA(1) = ( DFLOAT(5) + DSQRT(DFLOAT(15)) ) / DFLOAT(60) +.1
      BETA(1) = DFLOAT(3) / DFLOAT(20)
      BETA(2) = (DFLOAT(30)*BETA(1) - DFLOAT(1)) /
     -               (DFLOAT(360)*BETA(1) - DFLOAT(30))
      TAU(2) = 0.99
      TAU(1) = Dsqrt( ( DFLOAT(2) - DFLOAT(5) * (TAU(2)**2) ) /
     -          ( DFLOAT(5) * (DFLOAT(1) - DFLOAT(6)*(TAU(2)**2)) ) )
      ALPHA(1) = BETA(1) + BETA(2)
      ALPHA(2) = BETA(1) * BETA(2)
C
      A0 = (ALPHA(2) - (ALPHA(1)/DFLOAT(2)) + (DFLOAT(1)/DFLOAT(24)))
     -      / ALPHA(2)
      A(1) = (DFLOAT(1) - DFLOAT(12) * BETA(2)) /
     -       (DFLOAT(24)*BETA(1) * (BETA(1)-BETA(2)) )
      A(2) = (DFLOAT(1) - DFLOAT(12) * BETA(1)) /
     -       (DFLOAT(24)*BETA(2) * (BETA(2)-BETA(1)) )
      B = (K**2) / ( (TAU(2)**2 - TAU(1)**2) * (BETA(1) - BETA(2)) )
      C(1) = B * ( BETA(1)/DFLOAT(12) - ALPHA(2) + TAU(1)**2*
     -             (ALPHA(1)/DFLOAT(2) - DFLOAT(1)/DFLOAT(24)
     -                         - BETA(1)/DFLOAT(2)) )
      C(2) = B * ( ALPHA(2) - BETA(2)/DFLOAT(12) - TAU(1)**2*
     -             (ALPHA(1)/DFLOAT(2) - DFLOAT(1)/DFLOAT(24)
     -                         - BETA(2)/DFLOAT(2)) )
      D(1) = B * ( -BETA(1)/DFLOAT(12) + ALPHA(2) + TAU(2)**2*
     -             (-ALPHA(1)/DFLOAT(2) + DFLOAT(1)/DFLOAT(24)
     -                         + BETA(1)/DFLOAT(2)) )
      D(2) = B * ( -ALPHA(2) + BETA(2)/DFLOAT(12) + TAU(2)**2*
     -             (ALPHA(1)/DFLOAT(2) - DFLOAT(1)/DFLOAT(24)
     -                         - BETA(2)/DFLOAT(2)) )

C
      CALL Clear_Vector (U, BIG_N)
      DO I = 1,BIG_N
        CALL Get_U_Exact (U1(I), (DFLOAT(I)*H), K)
      END DO
C
      RETURN
      END
C
C***********************************************************************C
      SUBROUTINE Clear_Matrix (A, N)
C-----------------------------------------------------------------------C
C                                                                       C
C     This procedure sets all entries in matrix A to (0.0).             C
```

```
C                                                                       C
C-----------------------------------------------------------------------C
C
      INTEGER  MAXBIG_N
      PARAMETER (MAXBIG_N = 350)
      REAL*8 A(MAXBIG_N,MAXBIG_N)
      INTEGER ROW, COL, N
C
      DO ROW = 1,N
        DO COL = 1,N
          A(ROW,COL) = (0.0)
        END DO
      END DO
      RETURN
      END
C
C***********************************************************************C
      SUBROUTINE CLEAR_VECTOR (A, N)
C-----------------------------------------------------------------------C
C                                                                       C
C     This procedure sets all entries in vector A to (0.0).             C
C                                                                       C
C-----------------------------------------------------------------------C
C
      INTEGER  MAXBIG_N
      PARAMETER (MAXBIG_N = 350)
      REAL*8 A(MAXBIG_N)
      INTEGER ROW, N
C
      DO ROW = 1,N
          A(ROW) = (0.0)
      END DO
      RETURN
      END
C
C***********************************************************************C
      SUBROUTINE Error (U,NORM,N,H)
C-----------------------------------------------------------------------C
C                                                                       C
C     This procedure figures the error between U and U_EXACT and        C
C       puts the answer in NORM.                                        C
C                                                                       C
C-----------------------------------------------------------------------C
C
      INTEGER  MAXBIG_N
      PARAMETER (MAXBIG_N = 350)
      REAL*8 U(MAXBIG_N), U_EXACT, NORM, TEMP, H, T
      INTEGER N, I
C
      NORM = (0.0)
      T = (1.0)
      NORM = 0.0
c      WRITE (75,*) ('U_EXACT IS')
      DO I = 1,N
        CALL Get_U_Exact (U_EXACT, (DFLOAT(I)*H), T)
1402    FORMAT (F36.33,I6,F37.33)
```

```
c         WRITE (75,1402)I*H,I,U_EXACT
C           TEMP = ((DREAL(U(I)) - DREAL(U_EXACT))**2 +
C     -             (DIMAG(U(I)) - DIMAG(U_EXACT))**2)
           TEMP = DABS(U(I) - U_EXACT)
           IF (TEMP.GT.NORM) NORM = TEMP
       END DO
C       NORM = (NORM) / DFLOAT(N)
       RETURN
       END
C*********************************************************************C
       SUBROUTINE Get_U_Exact (U,X,T)
C---------------------------------------------------------------------C
C                                                                     C
C     This subroutine returns the value of u(x,t).                    C
C                                                                     C
C---------------------------------------------------------------------C
C
       REAL*8 X, U, T, ALPHA
C
       ALPHA = 4.0
       U = T**ALPHA * X * ((1.0) - X)
       RETURN
       END
```

```
      PROGRAM hypworker
C*********************************************************************C
C                                                                     C
C     This program solves the hyperbolic PDE:  Utt - Uxx = f(x,t).    C
C     This is a 4th order method which is 6th order for certain       C
C       choices of BETA and TAU.                                      C
C     This program uses the partial fractions approach and is the     C
C       parallelized method using PVM.  This program also requires    C
C       hyp-master.                                                   C
C                                                                     C
C*********************************************************************C
      INCLUDE 'pvm3/include/fpvm3.h'
C
      INTEGER BIG_N, N, MAXBIG_N
      PARAMETER ( MAXBIG_N = 350)
      INTEGER mytid, info, mastertid, msgtype
      REAL*8  Y(MAXBIG_N), U1(MAXBIG_N), H, K, A, C, D,
     -        TAU(2), RIGHT(MAXBIG_N), LEFT(MAXBIG_N,MAXBIG_N),
     -        LOWER(MAXBIG_N,MAXBIG_N), UPPER(MAXBIG_N,MAXBIG_N),
     -        Z(MAXBIG_N), BETA
C
C     * Enroll in PVM *
      CALL pvmfmytid( mytid )
C     * Get the master's task id *
      CALL pvmfparent( mastertid )
C
C     * Receive info needed from master *
      msgtype = 1
      CALL pvmfrecv( mastertid, msgtype, info )
      CALL pvmfunpack( 3, BIG_N, 1, 1, info )
      CALL pvmfunpack( 6, U1, MAXBIG_N, 1, info )
      CALL pvmfunpack( 6, K, 1, 1, info )
      CALL pvmfunpack( 6, H, 1, 1, info )
      CALL pvmfunpack( 6, A, 1, 1, info )
      CALL pvmfunpack( 6, C, 1, 1, info )
      CALL pvmfunpack( 6, D, 1, 1, info )
      CALL pvmfunpack( 6, TAU, 2, 1, info )
      CALL pvmfunpack( 6, BETA, 1, 1, info )
      CALL pvmfunpack( 3, N, 1, 1, info )
C
C       * Work involved *
      CALL Get_Right_Hand_Side (RIGHT,U1,N,BIG_N,K,H,TAU,A,C,D)
      CALL Get_Left_Hand_Side (LEFT,BIG_N,K,H,BETA)
      CALL Factor (LEFT, LOWER, UPPER, BIG_N)
      CALL LSolve (LOWER, Z, RIGHT, BIG_N)
      CALL USolve (UPPER, Y, Z, BIG_N)
C
C
C       * Send back results *
      msgtype = 7
      CALL pvmfinitsend( PVMDEFAULT, info )
      CALL pvmfpack( 6, Y, MAXBIG_N, 1, info )
      CALL pvmfsend( mastertid, msgtype, info )

      CALL pvmfexit()
      STOP
```

```
      END
C
C*********************************************************************C
      SUBROUTINE Get_Right_Hand_Side (R,U1,N,BIG_N,K,H,TAU,A,C,D)
C---------------------------------------------------------------------C
C                                                                     C
C     This procedure figures up the Right Hand side of the equation   C
C       used in the numerical scheme.  The equation is given in my    C
C       thesis.                                                       C
C     It figures up RIGHT depending what row L is currently at and then C
C        RIGHT = 2A U1 + K^2 D Fh(TAU(1),K) + K^2 C Fh(TAU(2),K)      C
C                                                                     C
C---------------------------------------------------------------------C
C
      INTEGER  MAXBIG_N
      PARAMETER (MAXBIG_N = 350)
      REAL*8 R(MAXBIG_N), U1(MAXBIG_N), K, H, TAU(2), Tn,
     -       F1(MAXBIG_N), F2(MAXBIG_N), F_PLUS, F_MINUS, T1, T2,
     -       A, C, D
      INTEGER N, BIG_N, L
C
      Tn = DFLOAT(N)*K
C
      T1 = Tn + TAU(1)*K
      T2 = Tn - TAU(1)*K
      DO L = 1,BIG_N
        CALL Get_F (F_PLUS,L,T1,H)
        CALL Get_F (F_MINUS,L,T2,H)
        F1(L) = F_PLUS + F_MINUS
      END DO
C
      T1 = Tn + TAU(2)*K
      T2 = Tn - TAU(2)*K
      DO L = 1,BIG_N
        CALL Get_F (F_PLUS,L,T1,H)
        CALL Get_F (F_MINUS,L,T2,H)
        F2(L) = F_PLUS + F_MINUS
      END DO
C
      DO L = 1,BIG_N
        R(L) = 2*A*U1(L) + D*F1(L) + C*F2(L)
      END DO
C
      RETURN
      END
C
C*********************************************************************C
      SUBROUTINE Get_F (F,I,T,H)
C---------------------------------------------------------------------C
C                                                                     C
C     This procedure gets the value of F(X,T) where F is the value to C
C       the right of the PDE, X = IxH, and T = T.                     C
C                                                                     C
C---------------------------------------------------------------------C

C
```

```
      REAL*8 H, T, F, ALPHA, X
      INTEGER I
C
      ALPHA = (4.0)
      X = DFLOAT(I) * H
      F = (T**(ALPHA-(2.0))) * (ALPHA*(ALPHA-1.0)*X*((1.0)-X)+
     -    (2.0)*(T**2))
C
      RETURN
      END
C
C*********************************************************************C
      SUBROUTINE Get_Left_Hand_Side (L,BIG_N,K,H,BETA)
C---------------------------------------------------------------------C
C                                                                     C
C     This procedure figures up the left side of the numerical scheme C
C       which happens to be (I-BETA K^2 Lh).  Where Lh is your        C
C       traditional finite difference matrix -DIAGONAL(-1 2 -1).      C
C                                                                     C
C---------------------------------------------------------------------C
C
      INTEGER  MAXBIG_N
      PARAMETER (MAXBIG_N = 350)
      REAL*8 L(MAXBIG_N,MAXBIG_N), H, K, BETA
      INTEGER  BIG_N, ROW
C
      CALL Clear_Matrix (L, BIG_N)
      IF (BIG_N.EQ.1) THEN
        ROW = 1
        L(ROW,ROW+1) = - BETA*(K**2)*(H**(-2.0))
        L(ROW,ROW) = (1.0) + BETA*(K**2)*2.0*(H**(-2.0))
      ELSE
        ROW = 1
        L(ROW,ROW+1) = - BETA*(K**2)*(H**(-2.0))
        L(ROW,ROW) = (1.0) + BETA*(K**2)*2.0*(H**(-2.0))
        ROW = BIG_N
        L(ROW,ROW) = (1.0) + BETA*(K**2)*2.0*(H**(-2.0))
        L(ROW,ROW-1) = - BETA*(K**2)*(H**(-2.0))
        DO ROW = 2,BIG_N-1
          L(ROW,ROW+1) = - BETA*(K**2)*(H**(-2.0))
          L(ROW,ROW) = (1.0) + BETA*(K**2)*2.0*(H**(-2.0))
          L(ROW,ROW-1) = - BETA*(K**2)*(H**(-2.0))
        END DO
      END IF
C
      RETURN
      END
C

C*********************************************************************C
      SUBROUTINE Clear_Matrix (A, N)
C---------------------------------------------------------------------C
C                                                                     C
C     This procedure sets all entries in matrix A to (0.0).           C
C                                                                     C
C---------------------------------------------------------------------C
```

```
C
      INTEGER  MAXBIG_N
      PARAMETER (MAXBIG_N = 350)
      REAL*8 A(MAXBIG_N,MAXBIG_N)
      INTEGER ROW, COL, N
C
      DO ROW = 1,N
        DO COL = 1,N
          A(ROW,COL) = (0.0)
        END DO
      END DO
      RETURN
      END
C
C*********************************************************************C
      SUBROUTINE CLEAR_VECTOR (A, N)
C---------------------------------------------------------------------C
C                                                                     C
C     This procedure sets all entries in vector A to (0.0).           C
C                                                                     C
C---------------------------------------------------------------------C
C
      INTEGER  MAXBIG_N
      PARAMETER (MAXBIG_N = 350)
      REAL*8 A(MAXBIG_N)
      INTEGER ROW, N
C
      DO ROW = 1,N
          A(ROW) = (0.0)
      END DO
      RETURN
      END
C
C*********************************************************************C
      SUBROUTINE FACTOR(A, L, U, N)
C---------------------------------------------------------------------C
C                                                                     C
C     This procedure factors A into L and U where L is a lower        C
C     triangular matrix and U is an upper triangular matrix.          C
C                                                                     C
C---------------------------------------------------------------------C
C
      INTEGER  MAXBIG_N
      PARAMETER (MAXBIG_N = 350)
      REAL*8 TEMP, A(MAXBIG_N,MAXBIG_N), L(MAXBIG_N,MAXBIG_N),
     -       U(MAXBIG_N,MAXBIG_N)
      INTEGER I, J, K, N
C
      DO I = 1,N
        L(I,I) = (1.0)
      END DO
C
      U(1,1) = A(1,1)
      IF (U(1,1).EQ.(0.0)) THEN
1100     FORMAT(' 1  FACTORIZATION IMPOSSIBLE!')
1350     FORMAT(' 2  FACTORIZATION IMPOSSIBLE!')
```

```
        WRITE (6,1100)
        STOP
      END IF
C
      DO I = 2,N
        U(1,I) = A(1,I) / L(1,1)
        L(I,1) = A(I,1) / U(1,1)
      END DO
C
      DO I = 2,N-1
        TEMP = (0.0)
        DO K = 1,I-1
          TEMP = TEMP + (L(I,K)*U(K,I))
        END DO
        U(I,I) = A(I,I) - TEMP
        IF (U(I,I).EQ.(0.0)) THEN
          WRITE (6,1350)
          STOP
        END IF
C
        DO J = I+1,N
          TEMP = (0.0)
          DO K = 1,I-1
            TEMP = TEMP + (L(I,K) * U(K,J))
          END DO
          U(I,J) = (A(I,J) - TEMP) / L(I,I)
C
          TEMP = (0.0)
          DO K = 1,I-1
            TEMP = TEMP + (L(J,K) * U(K,I))
          END DO
          L(J,I) = (A(J,I) - TEMP) / U(I,I)
        END DO
      END DO
C
      TEMP = (0.0)
      DO K = 1,N-1
        TEMP = TEMP + (L(N,K) * U(K,N))
      END DO
      U(N,N) = A(N,N) - TEMP
      RETURN
      END
C
C*********************************************************************C
      SUBROUTINE TRANSPOSE (A,TRANS,N)
C---------------------------------------------------------------------C
C                                                                     C
C     This procedure figures the transpose of matrix A and puts the   C
C     answer in TRANS.                                                C
C                                                                     C
C---------------------------------------------------------------------C
C
      INTEGER  MAXBIG_N
      PARAMETER (MAXBIG_N = 350)
      REAL*8 A(MAXBIG_N,MAXBIG_N), TRANS(MAXBIG_N,MAXBIG_N)
      INTEGER N, ROW, COL
```

```
C
      DO ROW = 1,N
        DO COL = 1,N
          TRANS(COL,ROW) = A(ROW,COL)
        END DO
      END DO
      RETURN
      END
C
C*********************************************************************C
      SUBROUTINE USOLVE(U, W, X, N)
C---------------------------------------------------------------------C
C                                                                     C
C     This procedure takes an upper triangular matrix, U, and solves  C
C     the equation UW=X for W.                                        C
C                                                                     C
C---------------------------------------------------------------------C
C
      INTEGER  MAXBIG_N
      PARAMETER (MAXBIG_N = 350)
      REAL*8 U(MAXBIG_N,MAXBIG_N), W(MAXBIG_N), X(MAXBIG_N), TEMP
      INTEGER I, J, K, N
C
      W(N) = X(N) / U(N,N)
C
      DO I = N - 1,1,-1
        DO J = 1,N
          TEMP = (0.0)
          DO K = I + 1,N
            TEMP = TEMP + (U(I,K)*W(K))
          END DO
          W(I) = (X(I) - TEMP) / U(I,I)
        END DO
      END DO
      RETURN
      END
C
C*********************************************************************C
      SUBROUTINE LSOLVE(L, M, W, N)
C---------------------------------------------------------------------C
C                                                                     C
C     This procedure takes a lower triangular matrix, L, and solves the C
C     equation LM=W for M.                                            C
C                                                                     C
C---------------------------------------------------------------------C
C
      INTEGER  MAXBIG_N
      PARAMETER (MAXBIG_N = 350)
      REAL*8 L(MAXBIG_N,MAXBIG_N), M(MAXBIG_N), W(MAXBIG_N), TEMP
      INTEGER I, J, K, N
C
      DO I = 1,N
        M(1) = W(1)
      END DO
C
      DO I = 2,N
```

```
      DO J = 1,N
        TEMP = (0.0)
        DO K = 1,I - 1
          TEMP = TEMP + (L(I,K)*M(K))
        END DO
        M(I) = (W(I) - TEMP)
      END DO
    END DO
    RETURN
    END
```

VITA

Kenneth William Fister was born in Lexington, Kentucky, on June 5, 1968. He lived in Georgetown, Kentucky, until after graduating from college. He graduated from Scott County High School, located in Georgetown, Kentucky, in May 1986. He then attended Georgetown College, also, located in Georgetown, Kentucky, where he graduated summa cum laude in May 1990 with a Bachelor of Science degree in mathematics and computer science. Presently he attends the University of Tennessee, Knoxville, where he will receive a Master of Science degree in mathematics in December 1994.