Symbolic and Numerical Computation for Artificial Intelligence

edited by

Bruce Randall Donald
Department of Computer Science
Cornell University, USA

Deepak Kapur
Department of Computer Science
State University of New York, USA

Joseph L. Mundy
AI Laboratory
GE Corporate R&D, Schenectady, USA

Academic Press
Harcourt Brace Jovanovich, Publishers
London San Diego New York
Boston Sydney Tokyo Toronto

ACADEMIC PRESS LIMITED
24–28 Oval Road
London NW1

US edition published by
ACADEMIC PRESS INC.
San Diego, CA 92101

This book is printed on acid-free paper

A catalogue record for this book is available from the British Library

ISBN 0-12-220535-9

Printed and Bound in Great Britain by
The University Press, Cambridge

Chapter 12

Symbolic and Parallel Adaptive Methods for Partial Differential Equations

Joseph E. Flaherty, Messaoud Benantar, Rupak Biswas[†]

Department of Computer Science

Rensselaer Polytechnic Institute

Troy, NY 12180

Peter K. Moore

Department of Mathematics

Tulane University

New Orleans, Louisiana 70118

In order to enhance the use of adaptive mesh refinement and order enrichment methods for partial differential equations by practising scientists and engineers, we have developed a symbolic interface to the numerical software. Partial differential equations and data are described in a natural high-level language. FORTRAN code, required by the various numerical procedures for evaluating functions, Jacobians, etc. is automatically generated and properties of the system, such as linearity and symmetry, are deduced.

Parallel procedures for adaptive techniques are of high interest given the need to solve problems of increasing complexity. We describe techniques for solving two-dimensional vector systems of elliptic and hyperbolic partial differential equations on shared-memory parallel computers. Linear algebraic systems resulting from the finite element discretization of an elliptic problem using a hierarchical piecewise polynomial basis on a finite-quadtree-structured mesh are solved by a conjugate gradient technique with a symmetric successive over-relaxation preconditioner. System assembly and solution are processed in parallel with computations scheduled on noncontiguous quadrants of the tree in order to minimize process synchronization. Coloring unstructured meshes that result from quadtrees is far simpler than coloring more general meshes, and we describe a linear time complexity coloring procedure that uses a maximum of six colors.

Hyperbolic systems are approximated by an explicit finite volume technique and solved by a recursive local mesh refinement procedure on a tree-structured grid. Computational procedures that sequentially traverse the tree while processing solutions on each grid in parallel, that process solutions at the same tree level in parallel, and that dynamically assign processors to nodes of the tree have been developed and applied to an example that illustrates their performance.

[†] This research was partially supported by the U. S. Air Force Office of Scientific Research, Air Force Systems Command, USAF, under Grant Number AFOSR-90-0194; by the SDIO/IST under management of the U. S. Army Research Office under Contract Number DAAL03-90-G-0096; and by the National Science Foundation under Grant Numbers CDA-8805910 and CCR-8920694.

1. Introduction

Partial differential equations that arise in scientific and engineering applications typically feature solutions that develop, evolve, and decay on diverse temporal and spatial scales. The Fokker-Planck equation of mathematical physics may be used to illustrate this phenomena. Perspective renditions of its solution u as a function of two spatial arguments x and y are shown at four times t in figure 1 (Moore and Flaherty, 1990). A single "spike" in the probability density arising from an initial Maxwell-Boltzmann distribution evolves into the two spikes shown as time progresses. Traditional fixed-step and fixed-order finite difference and finite element techniques for solving such problems would either require excessive computing resources or fail to adequately resolve non-uniform behavior. By automatically refining, coarsening, or relocating meshes and by varying the order of numerical accuracy, adaptive methods offer greater efficiency, reliability, and robustness.

Adaptive software for ordinary differential equations has existed for some time and procedures that vary both mesh spacing and order of accuracy are in common use for both initial (Gear, 1971) and boundary (Ascher *et al.*, 1988) value problems. Only very special problems would warrant hand-coding a method. The situation is far more difficult for partial differential equations due to the great diversity of phenomena that can occur; however, some production-ready adaptive software has appeared (Bank, 1990) for elliptic problems. The state-of-the-art for transient problems lags behind that for elliptic systems but some projects are under way (Flaherty *et al.*, 1989). Adaptive strategies will either have to be retrofitted into an existing software system for solving partial differential equations, or have to be coupled with pre- and post-processing software tools before widespread use occurs. A complete adaptive package that is, for example, intended for use by individuals who have limited experience with scientific computing would have to contain the following sub-systems:

1 *a computer algebra interface* so that partial differential equations, boundary conditions, etc. may be described in relatively natural terms;
2 *a geometric modeling interface* for describing the computational domain Ω;
3 *an automatic discretization interface* that can introduce a computational mesh on Ω;
4 *solution procedures* to solve the partial differential equations;
5 *error estimation procedures* that provide local and/or global measures of solution accuracy;
6 *enrichment strategies* that improve solution resolution where needed by refining the mesh, altering the solution procedure, etc.;
7 *vector or parallel solution capabilities* for increased performance; and
8 *visualization tools* to analyze and interpret the results.

Of this group, we briefly review adaptive enrichment strategies and error estimation in section 2, describe a symbolic computational interface using MAPLE in section 3, and focus on parallel solution procedures for elliptic and hyperbolic systems in sections 4 and 5, respectively.

Parallel procedures are becoming increasingly important both as hardware systems become available and as problem complexity increases. The efficiency afforded by adaptive strategies, furthermore, cannot be ignored in a parallel computational environment

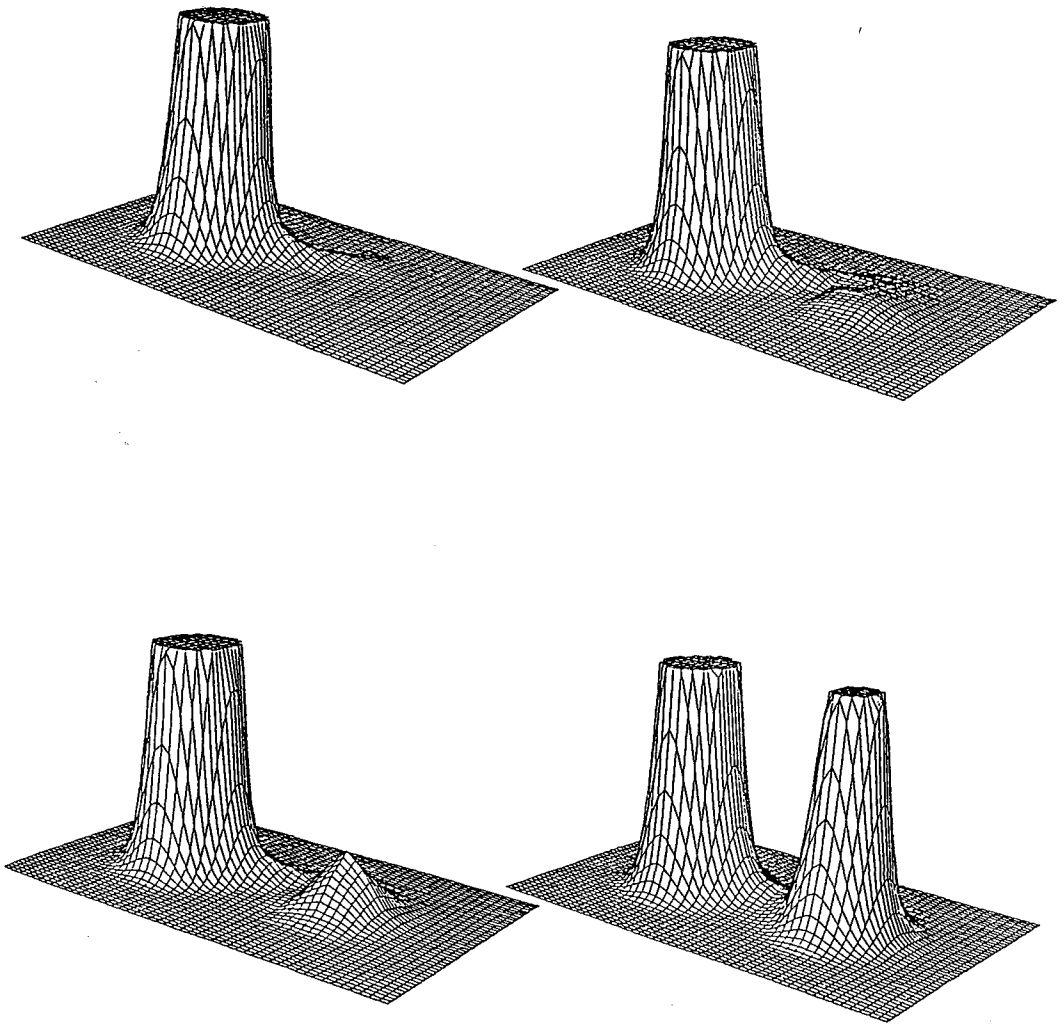

Figure 1. Solution $u(x, y, t)$ of the Fokker-Planck equation (Moore and Flaherty, 1990) at times $t = 4$ (upper left), 10 (upper right), 20 (lower left), and 100 (lower right). Solutions having magnitudes greater than 0.1 have been omitted in order to emphasize fine-scale structure.

```
procedure adaptive_PDE_solve ( tol : real );
        begin
                Select an approximation space;
                repeat
                        Compute a solution;
                        Estimate the error of the solution;
                        if error > tol then Enrich the approximation space
                until error ≤ tol
        end {adaptive_PDE_solve};
```

Figure 2. Top-level structure of an adaptive procedure for solving partial differential equations.

since the demand to model nature more accurately is always beyond hardware capabilities. Fortunately, adaptive software utilizes hierarchical (tree) data structures that have many embedded parallel constructs. "Coloring" the elements of a mesh so as to avoid memory contention on a shared-memory computer is far simpler when an underlying tree structure is present, than for more general unstructured grids. The six-color procedure described in section 4 for the finite element solution of elliptic problems on quadtree-structured grids displays a high degree of parallelism when piecewise linear approximations are used. The same procedure, unfortunately, does not do as well with higher-order piecewise polynomial approximations; however, an element edge coloring procedure may improve performance.

Transient hyperbolic problems may generally be solved using explicit numerical techniques which greatly simplify processor communication. Experiments, reported in section 5, with a variety of tree traversal strategies on an adaptive mesh refinement finite difference scheme for hyperbolic systems (Arney and Flaherty, 1990), indicate that the dynamic load balancing scheme of assigning grid-vertex computations at a given tree level to processors as they become available, provided the best parallel performance. Static load balancing strategies, that either traverse the tree of grids serially while processing solutions on each grid in parallel, or traverse the tree in parallel while processing solutions on grids at the same tree level, are also discussed in section 5. These alternatives to dynamic processor assignment may provide better performance on hierarchical memory computers.

2. Adaptive Procedures

A skeleton of a generic adaptive procedure is depicted in figure 2. An initial crude approximate solution generated on a coarse mesh with, perhaps, a low-order finite difference or finite element method, is enriched until a prescribed accuracy level is attained. Basic enrichment strategies, termed h-, p-, and r-refinement, may be used alone or in combination. With an h-refinement strategy, the computational mesh is refined or coarsened in regions of Ω that require more or less resolution (Bank, 1990; Arney and Flaherty, 1990). Sample meshes (Moore and Flaherty, 1990), used in conjunction with the adaptive h-refinement solution of the aforementioned Fokker-Planck equation, are shown in figure 3. Observe that (i) fine meshes are only introduced beneath the spikes in the solution, (ii) the problem is solved on successively smaller domains at each iteration of the solution procedure, and (iii) fine meshes are created and destroyed as the solution evolves. The order of accuracy of a finite difference or finite element solution scheme is varied

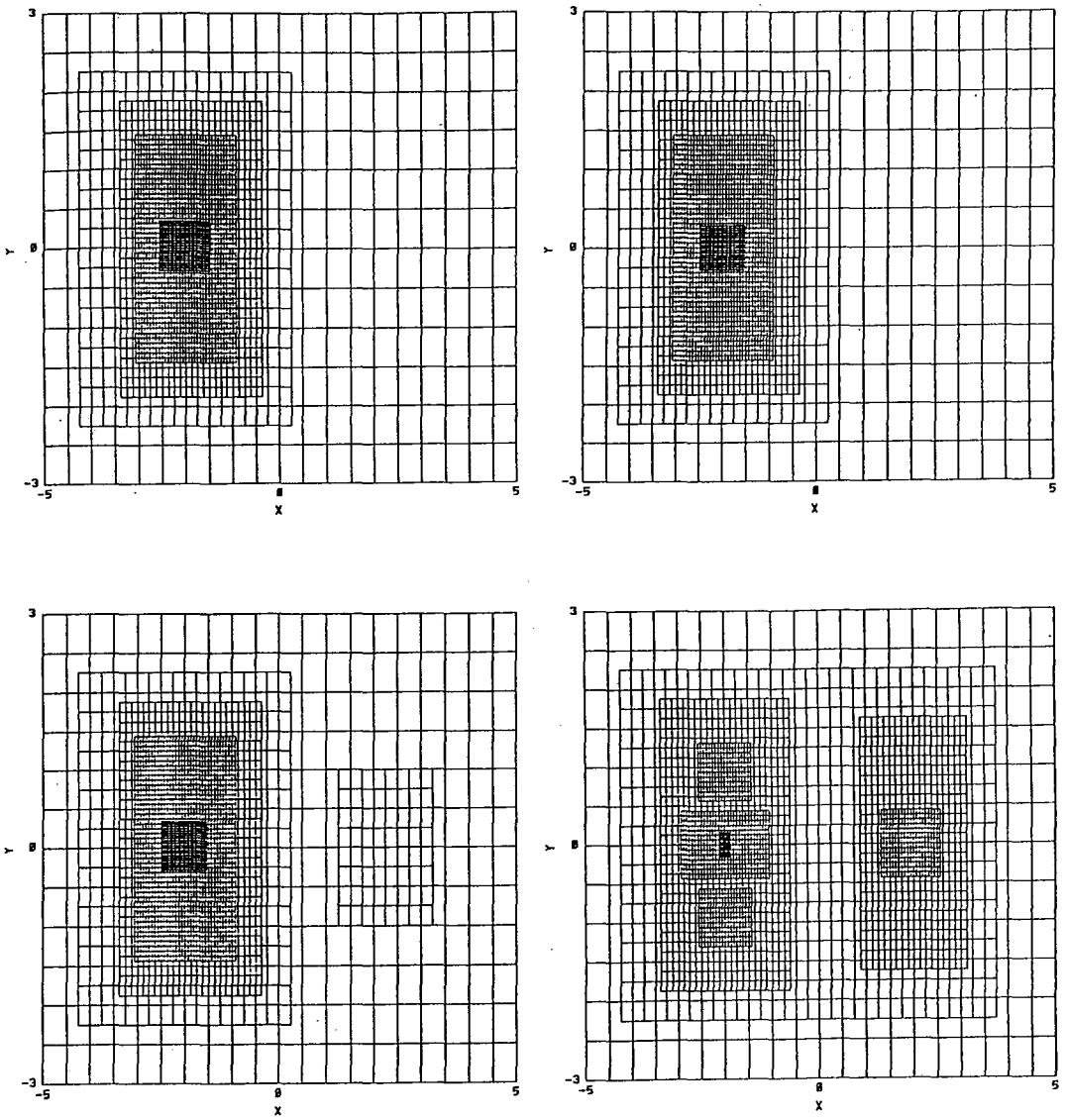

Figure 3. Sequence of adaptive meshes (Moore and Flaherty, 1990) at $t = 4$ (upper left), 10 (upper right), 20 (lower left), and 100 (lower right) used to obtain the solutions shown in figure 1.

in different regions of the problem domain, while the mesh is held fixed when using a p-refinement enrichment scheme (Babuska *et al.*, 1981). With r-refinement, a fixed topology mesh is redistributed so as to concentrate it in regions requiring more resolution or to follow dynamic phenomena (Arney and Flaherty, 1986).

Combining enrichment strategies can often produce dramatic increases in efficiency. The particular combination of h- and p-refinement, for example, has been shown to yield exponential convergence rates in certain situations (Babuska and Rank, 1986). While no combination is likely to exceed this dramatic convergence rate, other heuristic approaches might be more appropriate in certain situations. This is especially so with transient problems where r-refinement in combination with h-refinement has worked well on problems featuring isolated dynamic structures (Arney and Flaherty, 1990, 1986; Adjerid and Flaherty, 1986). Problems with discontinuous solutions, such as shock waves, might best use h- or r-refinement in the vicinity of the discontinuity while using p-refinement in regions where solutions are smooth. Research on such strategies is just beginning and there is much more to be learned.

Enrichment indicators, which are frequently estimates of the local discretization error of the numerical scheme, are used to control the adaptive process. Resources (finer meshes, higher-order methods, etc.) are introduced in regions having large enrichment indicators and deleted from regions where indicators are low. Using estimates of the discretization error as enrichment indicators also enables the calculation of local and global accuracy measures which should become a standard part of every scientific computation. Assignment of resources based on local error information seems natural; however, the optimality of this strategy can rarely be established.

Estimates of the local discretization error are typically obtained by using either h- or p-refinement. Thus, one uses the difference between solutions computed either on two meshes or with two distinct orders of accuracy to furnish an error estimate. Both approaches provide accurate assessments of errors in many cases; however, used as is, they are very expensive. Costs to obtain error estimates could be as much as four times the solution cost for a two-dimensional problem. Special "superconvergence" points, where solutions converge faster than they do globally, can be used to significantly reduce computational cost (Adjerid and Flaherty, 1986). Babuska and Yu (1990) discovered a dichotomy between the errors of finite element solutions obtained using odd or even piecewise polynomial bases. Dominant errors of odd-order approximations occur near element edges, while those of even-order approximations occur in element interiors. Thus, error estimates can be obtained by neglecting errors in element interiors or on element edges for odd- or even-order approximations, respectively. Under certain limiting hypotheses, Babuska and Yu (1990) proved that error estimates computed in this manner are asymptotically exact, as mesh spacing tends to zero for linear elliptic problems. Adjerid *et al.* (1990) established similar methods and analyses for parabolic systems.

3. Symbolic Problem Description

Consider the nonlinear second-order vector initial-boundary value problem

$$\mathbf{M}(\mathbf{x},t)\mathbf{u}_t + \mathbf{f}(\mathbf{x},t,\mathbf{u},\nabla\mathbf{u}) = \nabla\cdot\mathbf{D}(\mathbf{x},t,\mathbf{u})\nabla\,\mathbf{u}, \quad \mathbf{x}\in\Omega, \quad t>0, \tag{3.1}$$

$$\mathbf{u} = \mathbf{u}^0, \quad \mathbf{x} \in \Omega \cup \partial\Omega, \quad t = 0, \tag{3.2}$$

$$u_i = c_i^E(\mathbf{x}, t), \quad \mathbf{x} \in \partial\Omega_i^E, \quad (\mathbf{D}\mathbf{u}_\nu)_i = c_i^N(\mathbf{x}, t), \quad \mathbf{x} \in \partial\Omega_i^N, \quad t > 0,$$
$$i = 1, 2, \ldots, m, \tag{3.3}$$

where $\mathbf{u}$, $\mathbf{f}$, $\mathbf{c}$, and $\mathbf{u^0}$ are m-vectors; $\mathbf{M}$ and $\mathbf{D}$ are $m \times m$ matrices; t denotes time; $\mathbf{x}$ is a position vector on the one-, two, or three-dimensional problem domain Ω; ∇ is the gradient or divergence operator; $\partial\Omega = \partial\Omega_i^E \cup \partial\Omega_i^N$, $i = 1, 2, ..., m$, is the boundary of Ω; and ν is a unit outer normal vector to $\partial\Omega$. Numerical software for solving systems having the generality of (3.1-3.3) do not exist; however, with some simplifying assumptions, (3.1-3.3) can be reduced to a more standard elliptic, parabolic, or hyperbolic form for which software exists. With $\mathbf{M} = \mathbf{0}$, $\mathbf{D}$ positive definite, and (3.2) ignored, for example, (3.1-3.3) will be elliptic. Similarly, (3.1-3.3) is either parabolic or hyperbolic, respectively, when $\mathbf{M}$ and $\mathbf{D}$ are positive definite or when $\mathbf{D} = \mathbf{0}$.

Numerical procedures for solving (3.1-3.3) require the translation of the equations and data from their familiar mathematical form to a potentially complex language required by the software. Mathematical entities, such as Jacobians, are unessential to a user of the numerical software but, nevertheless, must be calculated for many implicit methods. Analytic Jacobians are often preferred to numerical approximations for reasons of efficiency and accuracy. Mistakes are easily made and difficult to find. We have developed a symbolic interface *pdefront* to our adaptive numerical software that alleviates these difficulties. Properties of particular systems can be deduced by *pdefront*, thereby reducing CPU time and storage of the numerical code. Our procedures are similar in spirit to those of Steinberg and Roache (1985, 1986, 1988a, 1988b) who have been using symbolic means to generate FORTRAN code for finite difference approximations for some time.

The *pdefront* system is written in a combination of C and MAPLE for Sun workstations. Equations are entered in a simple scalar mode. Users may either select problem-specific names for variables or use the default choices t, x, y, z, $u[1], u[2], ..., u[m]$. FORTRAN 77 subroutines are automatically generated for evaluating the functions $\mathbf{f}$, $\mathbf{D}$, etc. and the initial and boundary data. The appropriate adaptive numerical codes may also be executed.

Let us illustrate a typical *pdefront* session with an example.

Example 1. Consider the one-dimensional reaction-diffusion problem (Adjerid and Flaherty, 1986).

$$c_t + Dce^{-d/T} = c_{xx}, \quad LT_t - aDce^{-d/T} = T_{xx}, \quad 0 < x < 1, \quad t > 0, \tag{3.4}$$

$$c(x, 0) = T(x, 0) = 1, \quad 0 < x < 1, \tag{3.5}$$

$$c_x(0, t) = T_x(0, t) = 0, \quad c(1, t) = T(1, t) = 1, \quad t > 0. \tag{3.6}$$

After declaring the system to be parabolic and defining the dependent variable names, constants and parameters are defined for this example as $a = 1$, $L = 0.9$, $d = 20$, and $D = 1/4e^d$. Equation (3.4) is entered in the manner shown in figure 4. The definitional language uses MAPLE-like conventions for functions and simple operators. For example, $dif(c, t)$ denotes c_t and $exp(-d/T)$ is $e^{-d/T}$. Any initial or boundary conditions

```
pde
(define domain) (define equation) (generate) (execute) (edit) (help) (manual) (typeset) (exit)
Entry Mode: Equation System
System Type : Parabolic Hyperbolic Elliptic
Solution Method : MOL LRM
Spatial Dimension : 1D 2D 3D
Output File: f

Enter the endpoints of the spatial domain
The left endpoint is :
0.0
The right endpoint is :
1.0
Enter the initial time :
0.0
Mesh definition completed, select another menu item
Enter the constants used in the equations in the form:
c:=number;
After the last constant hit return
a:=1.0;
L:=0.9;
d:=20.0;
D:=exp(d)/4.0;


The default names of the independent variables are: t,x
Enter your choice of names in the same order as above separated by commas
or choose the default by simply typing <Return>

The default names of the dependent variables are: u[1],u[2],u[3],...
Enter your choice of names separated by commas or select the default names by typing <return>
c,T
Enter the equations in the equation system mode
End each equation with a ';'
After the last equation hit return
dif(c,t)+D*c*exp(-d/T)=dif(dif(c,x),x);
L*dif(T,t)-a*D*c*exp(-d/T)=dif(dif(T,x),x);


Enter the initial conditions in the form u[i]=f(x);
After the last initial condition hit return
c=1.0;
T=1.0;

Enter the boundary conditions for the boundary x = 0.000000e+00
in the form a*u[i]+b*dif(u[i],x)=f(t);
After the last boundary condition hit return
dif(c,x)=0.0;
Enter the time interval over which this boundary condition holds in the form
t1 < t < t2; where either inequality is optional
0<t;
```

Figure 4. Sample *pdefront* session for the reaction-diffusion system of Example 1.

```
SUBROUTINE FU(X,T,U,UX,D,NU)
IMPLICIT REAL*8 (A-H,O-Z)
D(1,1) = 0.1212912989D9*DEXP(-0.200D2/U(2))
D(1,2) = 0.242582597800D10*U(1)/U(2)**2*DEXP(-2.00D2/U(2))
D(2,1) = -0.1212912989D9*DEXP(-0.200D2/U(2))
D(2,2) = -0.242582597800D10*U(1)/U(2)**2*DEXP(-2.00D2/U(2))
RETURN
END
```

Figure 5. FORTRAN code generated for the Jacobian $\mathbf{f_u}(\mathbf{x}, t, \mathbf{u}, \nabla, \mathbf{u})$ of Example 1.

are entered subsequent to entering the differential equation. The boundary $\partial\Omega$ can be segmented in time so that discontinuous temporal boundary data can be prescribed. At present, there is no interface to geometrical modeling or mesh generation software, so Ω is restricted to be rectangular. FORTRAN subroutines are generated for all functions, Jacobians, and initial and boundary conditions that are required by the particular numerical software package once the problem-specific data has been entered. FORTRAN generated code for the Jacobian $\mathbf{f_u}$, which is required by all of our procedures for solving parabolic problems, is shown in figure 5.

Implicit finite difference or finite element discretizations of (3.1-3.3) lead to nonlinear algebraic systems which we solve by a modified Newton method. Until recently, our coding treated the algebraic problem as being nonlinear even when (3.1-3.3) were linear. This generally results in some loss of efficiency. For example, if a direct linear equation solver were being used in conjunction with the Newton iteration, then the modified Newton method would converge in one step. However, such convergence would not be detected until a second right side of the Newton system was generated. If linearity were known *a priori*, then this additional and possibly costly assembly step could be avoided. We automatically detect linearity by using capabilities within MAPLE. For example, the partial differential equation (3.1) is linear if both $\mathbf{f}(\mathbf{x}, t, \mathbf{u}, \nabla\mathbf{u})$ and $\mathbf{D}(\mathbf{x}, t, \mathbf{u})$ are linear functions of $\mathbf{u}$ and $\nabla\mathbf{u}$. Our algorithm declares (3.1) to be linear when (i) the dependent variable $\mathbf{u}$ is not an argument of nonlinear functions like sine and exp, and (ii) there are no combinations of dependent variables in algebraic expressions. Some linear functions may be declared as being nonlinear with this technique, but this inefficiency should occur infrequently.

Knowledge of a symmetric discrete operator could be used to save storage, reduce computing time, and create additional solution possibilities. Assuming an implicit temporal integration method, the *pdefront* software will determine the linearized operator resulting from the finite difference or finite element discretization of (3.1-3.3) and the use of Newton iteration to be symmetric if (i) $\mathbf{f}$ is independent of $\mathbf{u_x}$ and $\mathbf{f_u}$ is symmetric and (ii) $\mathbf{D}$ is independent of $\mathbf{u}$ and is symmetric. Once again, some symmetric Jacobians may be classified as being nonsymmetric by this test, but this should occur rarely. (The actual *pdefront* system permits separate diffusion matrices $\mathbf{D_1}$ and $\mathbf{D_2}$ to multiply the x and y components of $\nabla\mathbf{u}$ in (3.1-3.3). In this case, the symmetry test is applied to each component.)

4. Parallel Solution of Linear Systems Using Quadtree Structures

With the goal of describing a strategy for solving linear algebraic systems resulting from the finite element discretization of elliptic or parabolic systems, let us simplify (3.1-3.3) to the following linear two-dimensional elliptic system:

$$-[\mathbf{D}(x,y)\mathbf{u}_x]_x - [\mathbf{D}(x,y)\mathbf{u}_y]_y + \mathbf{Q}(x,y)\mathbf{u} = \mathbf{f}(x,y), \quad (x,y) \in \Omega, \tag{4.1}$$

$$u_i = c_i^E(x,y), \quad (x,y) \in \partial\Omega_i^E, \quad (\mathbf{D}\mathbf{u}_\nu)_i = c_i^N(x,y), \quad (x,y) \in \partial\Omega_i^N,$$

$$i = 1, 2, \ldots, m. \tag{4.2}$$

The diffusion $\mathbf{D}$ and source $\mathbf{Q}$ are, respectively, positive definite and positive semi-definite $m \times m$ matrices.

The Galerkin form of (4.1-4.3) consists of determining $\mathbf{u} \in H_E^1$ satisfying

$$A(\mathbf{v},\mathbf{u}) + (\mathbf{v},\mathbf{f}) = \sum_{i=1}^{m} \int_{\partial\Omega_i^N} \nu_i c_i^N \, ds, \quad \forall \mathbf{v} \in H_0^1, \tag{4.3}$$

where

$$A(\mathbf{v},\mathbf{u}) = \int_\Omega [\mathbf{v}_x^T \mathbf{D}\mathbf{u}_x + \mathbf{v}_y^T \mathbf{D}\mathbf{u}_y + \mathbf{v}^T \mathbf{Q}\mathbf{u}] \, dxdy, \quad (\mathbf{v},\mathbf{u}) = \int_\Omega \mathbf{v}^T \mathbf{u} \, dxdy. \tag{4.4}$$

As usual, the Sobolev space H^1 consists of functions having first partial derivatives in L^2. The subscripts E and 0 further restrict functions to satisfy the essential boundary conditions (4.2 & 4.3) and trivial versions of (4.2 & 4.3), respectively. Finite element solutions of (4.4 & 4.5) are constructed by approximating H^1 by a finite dimensional subspace $S^{N,p}$ and determining $\mathbf{U} \in S_E^{N,p}$ such that

$$A(\mathbf{V},\mathbf{U}) + (\mathbf{V},\mathbf{f}) = \sum_{i=1}^{m} \int_{\partial\Omega_i^N} V_i c_i^N, ds, \quad \forall \mathbf{V} \in S_0^{N,p}. \tag{4.5}$$

Selecting $S^{N,p}$ as a space of continuous piecewise $p\,th$-degree hierarchical polynomials (Szabo and Babuska, 1990) with respect to the partition of Ω into triangular finite elements described in section 4.1, substituting these approximations into (4.6), and evaluating the integrals by quadrature rules yields a sparse, symmetric, positive-definite, N-dimensional linear system of the form

$$\mathbf{K}\mathbf{X} = \mathbf{b}, \tag{4.6}$$

where $\mathbf{X}$ is an N-vector of Galerkin coordinates.

4.1. Finite Quadtree Mesh Structure

Meshes of triangular or quadrilateral elements are created automatically on Ω by using the *finite quadtree* procedure (Baehmann *et al.*, 1987). With this technique, Ω is embedded in a square "universe" that may be recursively quartered to create a set of disjoint

squares called *quadrants*. Data associated with quadrants is managed using a hierarchical tree structure with the original square universe regarded as the root and with smaller quadrants created by subdivision regarded as offspring of larger ones. Quadrants intersecting $\partial\Omega$ are recursively quartered until a prescribed spatial resolution of Ω has been obtained. At this stage, quadrants that are leaf nodes of the tree and intersect $\Omega \cup \partial\Omega$ are further divided into small sets of triangular or quadrilateral elements. Severe mesh gradation is avoided by imposing a maximal one-level difference between quadrants sharing a common edge. This implies a maximal two-level difference between quadrants sharing a common vertex. A final "smoothing" of the triangular or quadrilateral mesh improves element shapes and further reduces mesh gradation near $\partial\Omega$.

A simple example involving a domain consisting of a rectangle and a quarter circle, as shown in figure 6, will illustrate the finite quadtree process. In the upper left portion of the figure, the square universe containing the problem domain is quartered creating the one-level tree structure shown at the upper right. Were this deemed to be a satisfactory geometrical resolution, a mesh of five triangles could be created. As shown, the triangular elements are associated with quadrants of the tree structure. In the lower portion of figure 6, the quadrant containing the circular arc has been quartered and the resulting quadrant intersecting the circular arc has been quartered again creating the three-level tree shown in the lower right portion of the figure. A triangular mesh generated on this tree structure is also shown.

Arbitrarily complex two-dimensional domains may be discretized in this manner and generally produce unstructured grids; however, the underlying tree of quadrants remains regular. Adaptive mesh refinement is easily accomplished by subdividing appropriate leaf-node quadrants and generating a new mesh of triangular or quadrilateral elements locally; thus, unifying the mesh generation and adaptive solution phases of the problem under a common tree data structure.

4.2. Linear System Solution Strategies

Preconditioned conjugate gradient (PCG) iteration is an efficient means of solving the linear algebraic systems (4.7) that result from the finite element discretization of self-adjoint elliptic partial differential systems (Axelsson and Barker, 1984). The key steps in the PCG procedure (Ortega, 1988) involve (i) matrix-vector multiplication of the form

$$\mathbf{q} = \mathbf{K}\mathbf{p} \tag{4.7}$$

and (ii) solving linear systems of the form

$$\bar{\mathbf{K}}\mathbf{d} = \mathbf{r}, \tag{4.8}$$

where $\mathbf{r}$ and $\mathbf{p}$ are the residual vector and conjugate search direction, respectively. The preconditioning matrix $\bar{\mathbf{K}}$ may be selected to reduce computational cost. The element-by-element (EBE) and symmetric successive over-relaxation (SSOR) preconditionings are in common use and seem appropriate for use with quadtree structured grids. The EBE preconditioning is an approximate factorization of the stiffness matrix $\mathbf{K}$ into a product of elemental matrices. If the grid has been "colored" so as to segregate non-contiguous elements, then (4.9) can be solved in parallel on elements having the same color. Since the matrix-vector multiplication (4.8) can also be performed in an element-

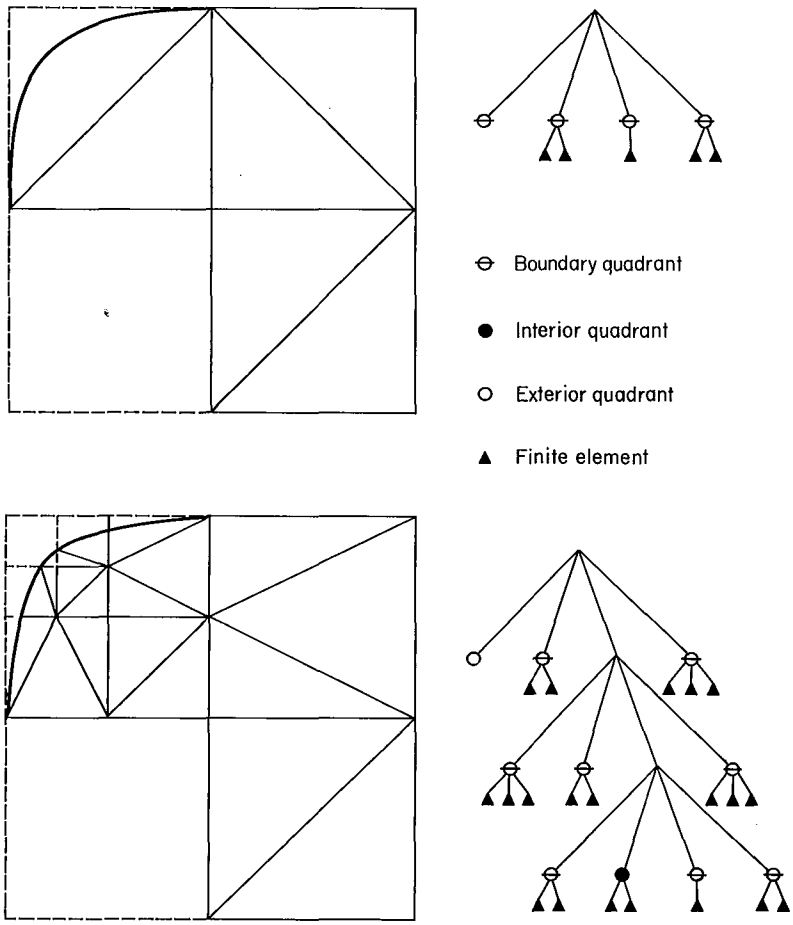

Figure 6. Finite quadtree mesh generation for a domain consisting of a rectangle and a quarter circle. One-level and three-level tree structures and their associated meshes of triangular elements are shown at the top and bottom of the figure, respectively.

by-element fashion, the entire PCG solution can be done in parallel on non-contiguous elements. While this simple approach has been used in several applications (Gustafsson and Lindskog, 1986; King and Sonnad, 1987; Nour-Omid and Parlett, 1985), we found the SSOR preconditioning to be more efficient in every instance (Benantar *et al.*, 1990) and, therefore, we shall not discuss EBE preconditionings further.

SOR and SSOR iteration has been used for the parallel solution of five-point difference approximations of Poisson's equation on rectangular meshes by numbering the discrete equations and unknowns in "checkerboard" order (Adams and Ortega, 1982). With this ordering, unknowns at "red" mesh points are only coupled to those at "black" mesh points and vice versa; thus, solutions at all red points can proceed in parallel. This step may be followed by a similar solution at all black points. Adams and Ortega (1982) extend this two-color ordering to multicolor orderings on rectangular grids using several common finite element and finite difference stencils. Multicolor orderings for unstructured meshes are far more difficult since nodal connectivity and difference stencils for high-degree polynomial approximations can be quite complex. The computational effort can be reduced when using quadtree-structured grids by considering multicolor orderings for block SSOR preconditionings at the quadrant level. To be specific, partition the stiffness matrix $\mathbf{K}$ by quadrants as

$$\mathbf{K} = \mathbf{D} - \mathbf{L} - \mathbf{L}^T \tag{4.9}$$

where

$$\mathbf{D} = \begin{bmatrix} \mathbf{K}_{1,1} & & & \\ & \mathbf{K}_{2,2} & & \\ & & \cdots & \\ & & & \mathbf{K}_{Q,Q} \end{bmatrix}, \quad \mathbf{L} = -\begin{bmatrix} 0 & & & \\ \mathbf{K}_{2,1} & 0 & & \\ & & \cdots & \\ \mathbf{K}_{Q,1} & \mathbf{K}_{Q,2} & & 0 \end{bmatrix}. \tag{4.10}$$

Nontrivial entries in a diagonal block $\mathbf{K}_{i,i}$ arise from Galerkin coordinates that are connected through the finite element basis to other unknowns in quadrant i. Nontrivial contributions to block $\mathbf{K}_{i,j}$ of the lower triangular matrix $\mathbf{L}$ arise when the support of the basis associated with a Galerkin coordinate in quadrant i intersects quadrant j.

Using an SSOR preconditioning, the solution of (4.9) would be computed according to the two-step procedure

$$\mathbf{X}^{n+1/2} = \omega(\mathbf{L}\mathbf{X}^{n+1/2} + \mathbf{L}^T\mathbf{X}^n + \mathbf{r}) + (1-\omega)\mathbf{X}^n, \tag{4.11}$$

$$\mathbf{X}^{n+1} = \omega(\mathbf{L}^T\mathbf{X}^{n+1} + \mathbf{L}\mathbf{X}^{n+1/2} + \mathbf{r}) + (1-\omega)\mathbf{X}^{n+1/2}, \quad n = 1, 2, \ldots, M. \tag{4.12}$$

Thus, each block SSOR iteration consists of two block SOR steps; one having the reverse ordering of the other. Typically, $M = 3$ SSOR steps are performed between each PCG step.

Suppose that the Q quadrants of a finite quadtree structure are separated into γ disjoint sets. Then, using the symmetric γ-color block SSOR ordering, we would sweep the quadrants in the order $C_1, C_2, \ldots, C_\gamma, C_\gamma, C_{\gamma-1}, \ldots, C_1$, where C_i is the set of quadrants having color i. Because quadrants rather than nodes are colored, a node can be connected to other nodes having the same color. Thus, the forward and backward SOR sweeps may differ for a color C_i, $i = 1, 2, \ldots, \gamma$. During an SOR sweep, unknowns lying on quadrant boundaries are updated as many times as the number of quadrants containing them.

4.3. Coloring Finite Quadtree Structures

Coloring the regular quadrants of a finite quadtree is far simpler than coloring the elements of a mesh. Differences in the small number of elements within quadrants having the same color may cause some load imbalance and this effect will have to be investigated. Naturally, coloring procedures that use the fewest colors increase data granularity and reduce the need for process synchronization. At the same time, the cost of the coloring algorithm should not be the dominant computational cost. With these views in mind, we developed an eight-color procedure that has linear time complexity (Benantar *et al.*, 1990). This procedure only required a simple breadth-first traversal of the quadtree, but performance never exceeded that of the six-color procedure which is described in the following paragraphs. Four-color procedures are undoubtedly possible, but we have not formulated any. Their complexity, unlike the eight- and six-color procedures, may be nonlinear.

With the aim of constructing a quadtree coloring procedure using a maximum of six colors, let us define a binary directed graph called a "quasi-binary tree" from the finite quadtree using the following recursive assertive algorithm.

1. The root of the quadtree corresponds to the root of the quasi-binary tree.
2. Every terminal quadrant is associated with a node in the quasi-binary tree; however, not every quasi-binary tree node must correspond to a quadrant.
3. In the planar representation of the quadtree, nodes across a common horizontal edge are connected in the quasi-binary tree.
4. When a quadrant is divided, its parent node in the quasi-binary tree becomes the root of a subtree.

Planar representations of simple quadtrees and their quasi-binary tree representations are illustrated in figure 7. The leftmost quadtree illustrates root-node and offspring construction of the quasi-binary tree. Connection of nodes across horizontal edges is shown with and without quadrant division in all three illustrations. Subtree definitions according to assertion 4 are shown in the center and rightmost quadtrees.

From figure 7, we see that column-order traversal of a finite quadtree is the depth-first traversal of its associated quasi-binary tree. Let us define six colors divided into three sets a, b, and c of two disjoint colors that alternate through the columns in a column-order traversal of the quadtree. Whenever left and right quasi-binary tree branches merge, column-order traversal continues using the color set associated with the left branch. Two of the three color streams, say a and b, are passed to a node of the quasi-binary tree. At each branching, the color stream a and the third color stream c are passed to the left offspring while the streams a and b in reverse order are passed to the right offspring. Additional details and a correctness proof of this algorithm will appear (Flaherty and Benantar, 1990).

Computational experiments (Benantar *et al.*, 1990) demonstrate the excellent parallelism that may be obtained by the six-color SSOR PCG procedure with piecewise linear finite element approximations. However, higher-order polynomial bases create additional possibilities for processor load imbalance with coloring at the quadrant level. Let us illustrate this with a simple problem that was solved using a 16-processor Sequent Balance 21000 shared-memory parallel computer. Parallelism on this system is supported through

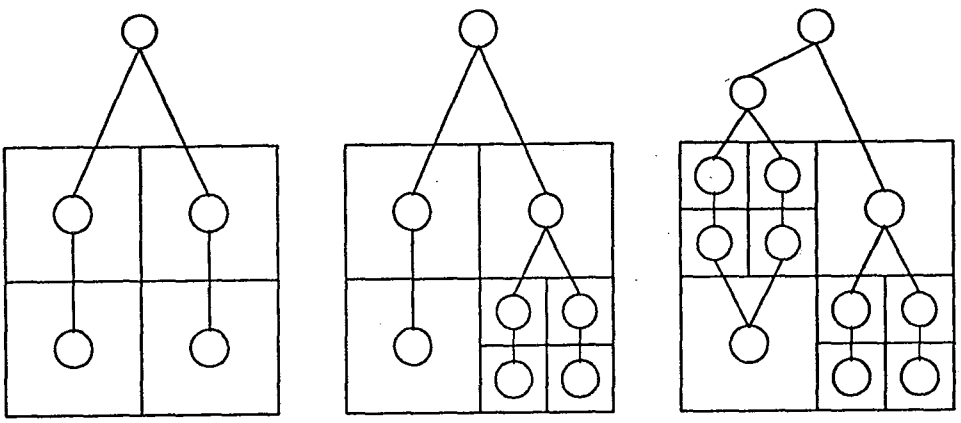

Figure 7. Planar representations of three quadtrees and their associated quasi-binary trees.

the use of a parallel programming library that permits the creation of parallel processes and enforces synchronization and communication using barriers and hardware locks.

Example 2. Consider the Dirichlet problem

$$u_{xx} + u_{yy} = f(x, y), \quad (x, y) \in \Omega, \tag{4.13}$$

$$u = 0, \quad (x, y) \in \partial\Omega, \tag{4.14}$$

with $\Omega = \{(x, y) | -3 < x, y < 3\}$. We solved this problem on a 400-element mesh using piecewise linear, quadratic, and cubic approximations. Adaptive p-refinement with the polynomial degree p restricted to be 1, 2, or 3 was also performed. Parallel speed up and processor idle time resulting from the need to synchronize at the completion of each color are shown in figure 8.

Parallel performance degrades as polynomial degree increases, with the adaptive strategy having the poorest performance. Adaptive algorithms typically have serial logic which limits speed up. Of course, speed up is not the only measure of complexity, and an adaptive solution strategy could require less CPU time to solve the problem to a given level of accuracy. Nevertheless, additional research is necessary to improve performance with high-order and adaptive strategies.

Using a hierarchical basis, all Galerkin coordinates for polynomial degrees higher than one are associated with mesh points that are either along element edges or within elements. Thus, the Galerkin coordinates for continuous piecewise linear approximations are the only ones associated with element vertices. Parallel performance could, therefore, be improved by coloring element edges rather than quadrants and we have designed a three-color procedure having linear time complexity to do this (Flaherty and Benantar, 1990). Since hierarchical bases add incremental corrections as the polynomial degree is increased, one could conceive an algorithm where quadrant coloring is used with the

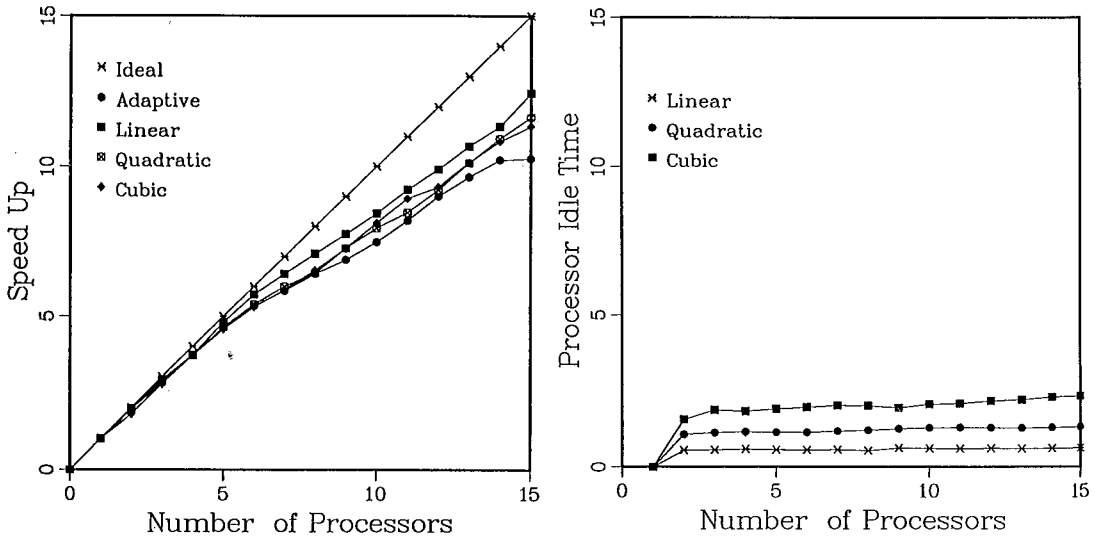

Figure 8. Parallel speed up (left) and processor idle time (right) for the finite element solution of Example 2 using piecewise linear, quadratic, and cubic approximations as well as adaptive p-refinement.

piecewise linear portion of the approximation, and edge coloring is used for higher-degree approximations.

5. Parallel Solution of Hyperbolic Systems

Let us once again reduce the general system (3.1-3.3) to the two-dimensional system of conservation laws

$$\mathbf{u}_t + \mathbf{f}_x(x, y, t, \mathbf{u}) + \mathbf{g}_y(x, y, t, \mathbf{u}) = 0, \quad (x, y) \in \Omega, \quad t > 0, \tag{5.1}$$

with the initial conditions

$$\mathbf{u}(x, y, 0) = \mathbf{u}^0(x, y), \quad (x, y) \in \Omega \cup \partial\Omega. \tag{5.2}$$

At present, Ω is also restricted to be rectangular.

Our research is based on an adaptive hr-refinement algorithm (Arney and Flaherty, 1990). We forgo mesh motion at present and briefly describe an h-refinement procedure that utilizes their strategy. The problem (5.1 & 5.2) is solved on a coarse rectangular "base

mesh" for a sequence of base-mesh time slices of duration Δt_n, $n = 0, 1, ...$, by an explicit finite difference, finite volume, or finite element scheme. For a base-mesh time step, say from t_n to $t_{n+1} = t_n + \Delta t_n$, a discrete solution is generated on the base mesh along with a set of local enrichment indicators which, in this case, are refinement indicators. Cells of the mesh where refinement indicators at t_{n+1} fail to satisfy a prescribed tolerance are identified and grouped into rectangular clusters. After ensuring that clusters have an adequate percentage of high-refinement-indicator cells, and subsequently enlarging the clusters by a one-element buffer to provide a transition between regions of high and low refinement indicators, cells of the base mesh are bisected in space and time to create finer meshes that are associated with each rectangular cluster. Local problems are solved on the finer meshes and the refinement procedure is repeated until refinement indicators satisfy the prescribed unit-step criteria. After finding an acceptable solution on the base mesh, the integration continues with, possibly, a new base-mesh time step Δt_{n+1}.

Data management involves the use of a tree structure with nodes of the tree corresponding to meshes at each refinement level for the current base-mesh time step. The base mesh is the root of the tree and finer grids are regarded as offspring of coarser ones. This structure is somewhat different from the tree of quadrants described in section 4.

With an aim of maintaining generality at the possible expense of accuracy and performance, we discretize (5.1 & 5.2) using the Richtmyer two-step version of the Lax-Wendroff method (Richtmyer and Morton, 1967), which we describe for a one-dimensional problem having no y dependence. Let us introduce a mesh on Ω having spacing $\Delta x_j = x_{j+1} - x_j$ and let the discrete approximation of $\mathbf{u}(x_j, t_n)$ be denoted as $\mathbf{U}_j^n$. Predicted solutions are generated at cell centers by the Lax-Friedrichs scheme, i.e.

$$\mathbf{U}_{j+1/2}^{n+1/2} = \frac{1}{2}\left(\mathbf{U}_{j+1}^n + \mathbf{U}_j^n\right) - \frac{\Delta t_n}{2\Delta x_j}\left(\mathbf{f}_{j+1}^n - \mathbf{f}_j^n\right). \tag{5.3}$$

This provisional solution is then corrected by the leapfrog scheme

$$\mathbf{U}_j^{n+1} = \mathbf{U}_j^n - \frac{2\Delta t_n}{\Delta x_j + \Delta x_{j-1}}\left(\mathbf{f}_{j+1/2}^{n+1/2} - \mathbf{f}_{j-1/2}^{n+1/2}\right). \tag{5.4}$$

Following Arney *et al.* (1990) and Arney and Flaherty (1990), refinement indicators are selected as estimates of the local discretization error obtained by Richardson's extrapolation (h-refinement) on a mesh having half the spatial and temporal spacing of the mesh used to generate the solution. Fine-mesh solutions generated as part of this error estimation process may subsequently be used on finer meshes when refinement is necessary. Initial and boundary data for refined meshes is determined by piecewise bilinear polynomial interpolation from acceptable solutions on the finest available meshes.

5.1. Parallel Computing Models

Parallel procedures are developed for the adaptive h-refinement solution scheme described above using a P-processor concurrent read exclusive write (CREW) shared-memory MIMD computer. We consider both static and dynamic strategies for balancing processor loading. As the names imply, with static load balancing, processors are assigned tasks *a priori* with the goal of having them all terminate at approximately the same time; whereas with dynamic load balancing, available processors are assigned tasks

from a task queue. Two possible static load balancing techniques come to mind: (i) serial depth-first traversal of the tree of grids with solutions on each grid generated in parallel and (ii) parallel generation of solutions on all grids that are at the same tree level. With the depth-first traversal procedure, each grid is divided into P subregions and a processor is assigned to each subregion. With the parallel tree traversal procedure, the P processors are distributed among all grids at a particular tree level so as to balance loading. Thus, parallelism occurs both within a grid and across the breadth of the tree with this strategy. In all cases, the parallel solution process proceeds from one base-mesh time step to the next.

Serial depth-first traversal of the tree leads to a highly structured algorithm that has a straight-forward design because the same procedure is used on all grids. Balancing processor loading on rectangular grids is nearly perfect with an explicit finite difference scheme like (5.3 & 5.4), and similar behavior could be expected for geometrically complex regions. Load imbalance occurs due to differences in the time required to compute initial data. Other than at $t = 0$, initial data is determined by interpolating solutions from the finest grid at the end of the previous base-mesh time step to the present grid. Tree traversal, required to determine the correct solution values for the interpolation, would generally take different times in different regions due to variations in tree depth. This defect might be remedied by using a different domain decomposition.

The serial depth-first traversal procedure becomes inefficient when P is of the order of the number of elements in a grid. This situation can be avoided by refining grids by more than a binary factor, thus maintaining a shallow tree depth. However, each cluster would then contain a greater percentage of low-refinement-indicator cells, thus lowering the (serial) efficiency of the procedure. The inefficiency cited here should not be a factor on data-parallel computers and the serial tree-traversal procedure might also be viable there.

A parallel tree-traversal procedure requires complex scheduling to assign processors to grids. One possibility is to estimate the work remaining to reduce error estimates to prescribed tolerances, and to assign processors to subgrids so as to balance this load. Were such a heuristic technique successful, the parallel tree traversal procedure would not degrade in efficiency when the number of elements on a grid is $O(P)$.

Consider a situation where Q processors are used to obtain a solution on a grid at tree level $l-1$ and suppose that refinement indicators dictate the creation of L grids $G_{l,i}$, $i = 1, 2, ..., L$, at level l. Further assume that

1 the prescribed local refinement tolerance at level $l-1$ is τ_{l-1};
2 the areas of $G_{l,i}$ are $M_{l,i}$, $i = 1, 2, ..., L$;
3 estimates $E_{l,i}$ of the discretization error are available for $G_{l,i}$, $i = 1, 2, ..., L$; and
4 the convergence rate of the numerical scheme is known as a function of the local mesh spacing.

The Richtmyer two-step scheme (5.3 & 5.4) has a quadratic convergence rate which we use to illustrate the load balancing technique.

In order to satisfy the prescribed accuracy criterion, $G_{l,i}$ should be refined by a factor of $(E_{l,i}/\tau_{l-1})^2$. The time step on $G_{l,i}$ must be reduced by a factor of $E_{l,i}/\tau_{l-1}$ in order to satisfy the Courant condition (Richtmyer and Morton, 1967). Hence, the expected work

$W_{l,i}$ to find an acceptable solution on $G_{l,i}$ is

$$W_{l,i} = M_{l,i} \left[\frac{E_{l,i}}{\tau_{l-1}} \right]^3 . \tag{5.5}$$

The Q available processors should be allocated so as to balance the time to complete the expected work on each of the L grids at level l. Thus, assign Q_i, $i = 1, 2, ..., L$, processors so that

$$\frac{W_{l,1}}{Q_1} = \frac{W_{l,2}}{Q_2} = \ldots = \frac{W_{l,L}}{Q_L}, \quad \sum_{i=1}^{L} Q_i = Q. \tag{5.6}$$

The quality of load balancing using this approach will depend on the accuracy of the discretization error estimate. Previous investigations (Arney and Flaherty, 1990; Arney *et al.*, 1990) revealed that error estimates were generally better than 80 percent of the actual error for a wide range of mesh spacings and problems. Equation (5.5) can be used to select refinement factors other than binary and, indeed, to select different refinement levels for different meshes at a given tree level. This consideration combined with over-refinement to a tolerance somewhat less than the prescribed tolerance should maintain a shallow tree depth and enhance parallelism at the expense of grid optimality.

Simple dynamic load balancing can take full advantage of the CREW shared-memory MIMD environment. One just maintains a queue of mesh points at a given tree level and computes solutions at these points as processors become available. Balancing processor loads on geometrically complex regions is as simple as on rectangular regions because mesh points are processed on a first-come-first-serve basis independently of the grid to which they belong. Non-uniformities in initial data also introduce no problems and neither does the relationship of P to the number of cells in a grid. Finally, complex processor scheduling based on accurate error estimates is avoided. This strategy, however, might not be appropriate for hierarchical or distributed memory computers.

5.2. An Adaptive H-Refinement Strategy

Binary refinement of space-time grids may be optimal in using the fewest mesh points; however, tree depth tends to be large and this introduces serial overhead into a parallel procedure. As previously suggested, serial overhead can be reduced by keeping tree depth shallow and to do this we perform M-ary instead of binary refinement. The value of M is chosen adaptively for different clusters so that the prescribed tolerance is satisfied after one refinement step. Thus, let τ_0 be a prescribed local discretization error tolerance and choose M for grid $G_{1,i}$ as the first even integer greater than $E_{1,i}/\tau_0$. Having a good *a priori* knowledge of the work required on each cluster, processors can be distributed among the grids according to (5.6) to effectively balance loading. Of course, the refinement tolerance may not be satisfied after performing one M-ary refinement. Should this occur, we perform additional levels of 2-ary refinement until tolerances are satisfied. The terms "binary" and "2-ary" refinement have been used to distinguish differences in our methods of checking the refinement condition. With binary refinement, the refinement condition is checked after each of the two finer time steps and with 2-ary refinement the condition is only checked after the second time step. As a result, the fine grids remain unchanged for both of the two finer time steps with 2-ary refinement.

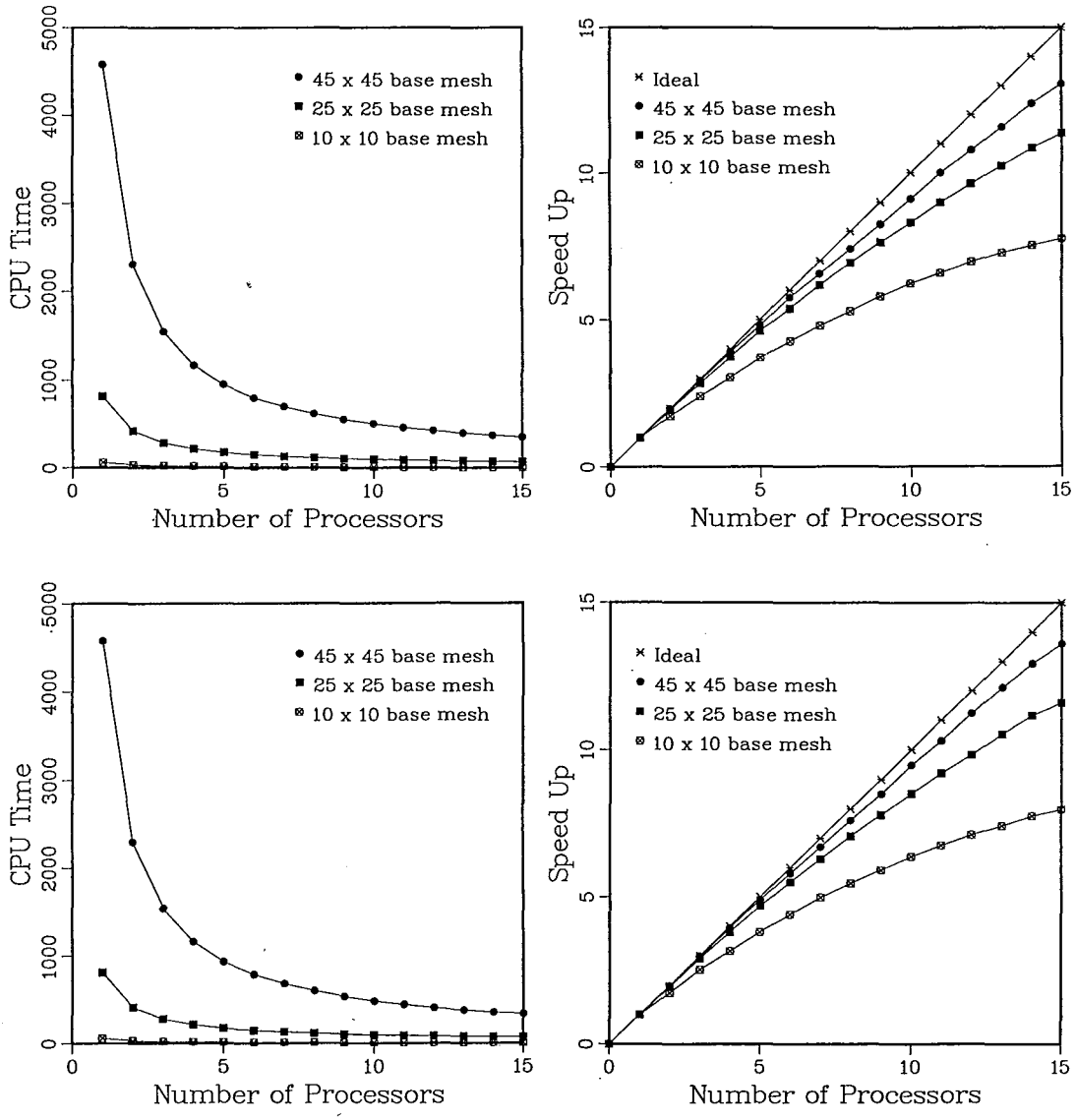

Figure 9. CPU time (left) and parallel speed up (right) for Example 3. Solutions are computed on uniform meshes without adaptivity using static (top) and dynamic (bottom) load balancing.

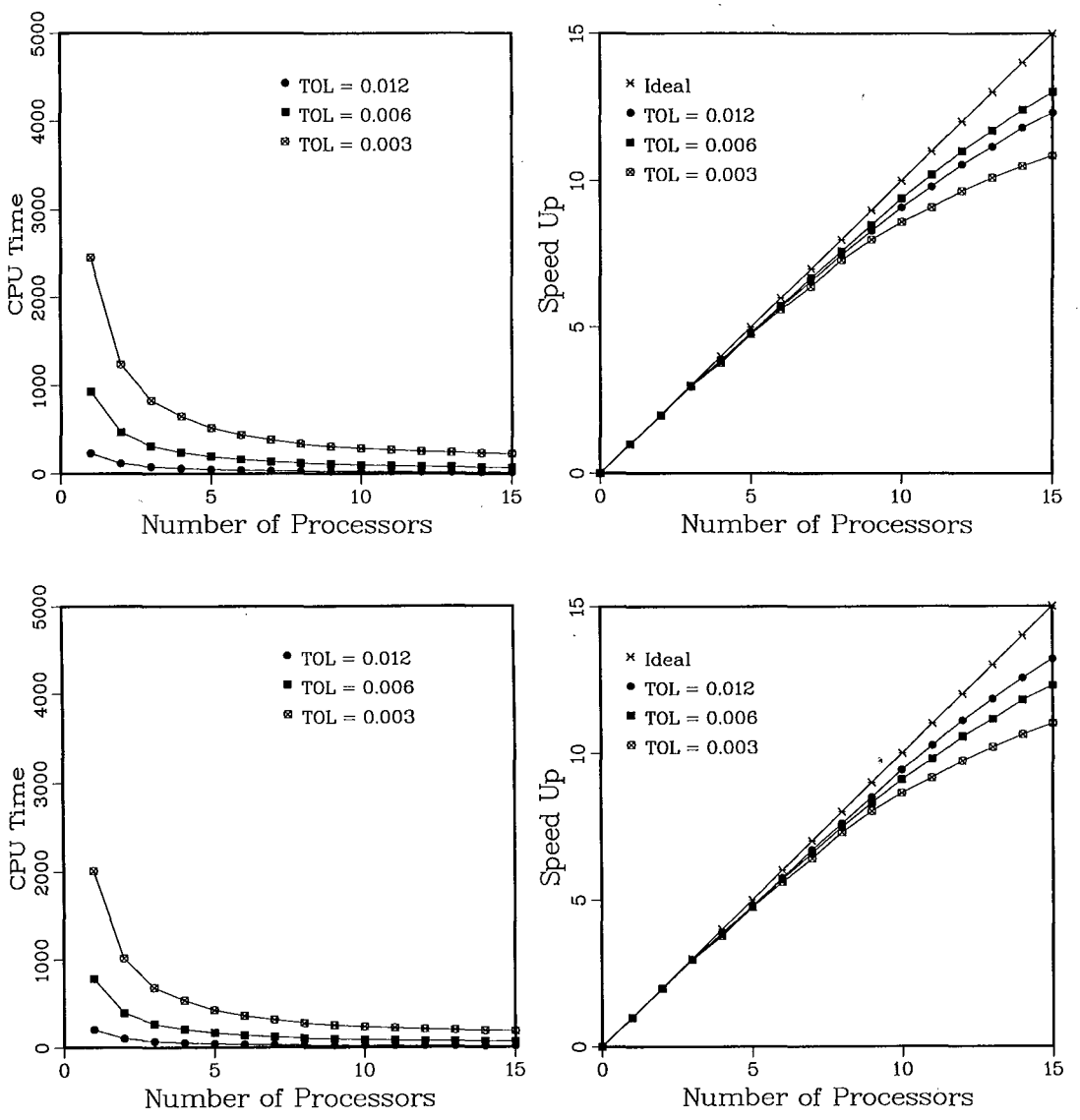

Figure 10. CPU time (left) and parallel speed up (right) for Example 3 using dynamic load balancing and adaptive h-refinement with either local binary refinement (top) or M-ary followed by 2-ary refinement (bottom).

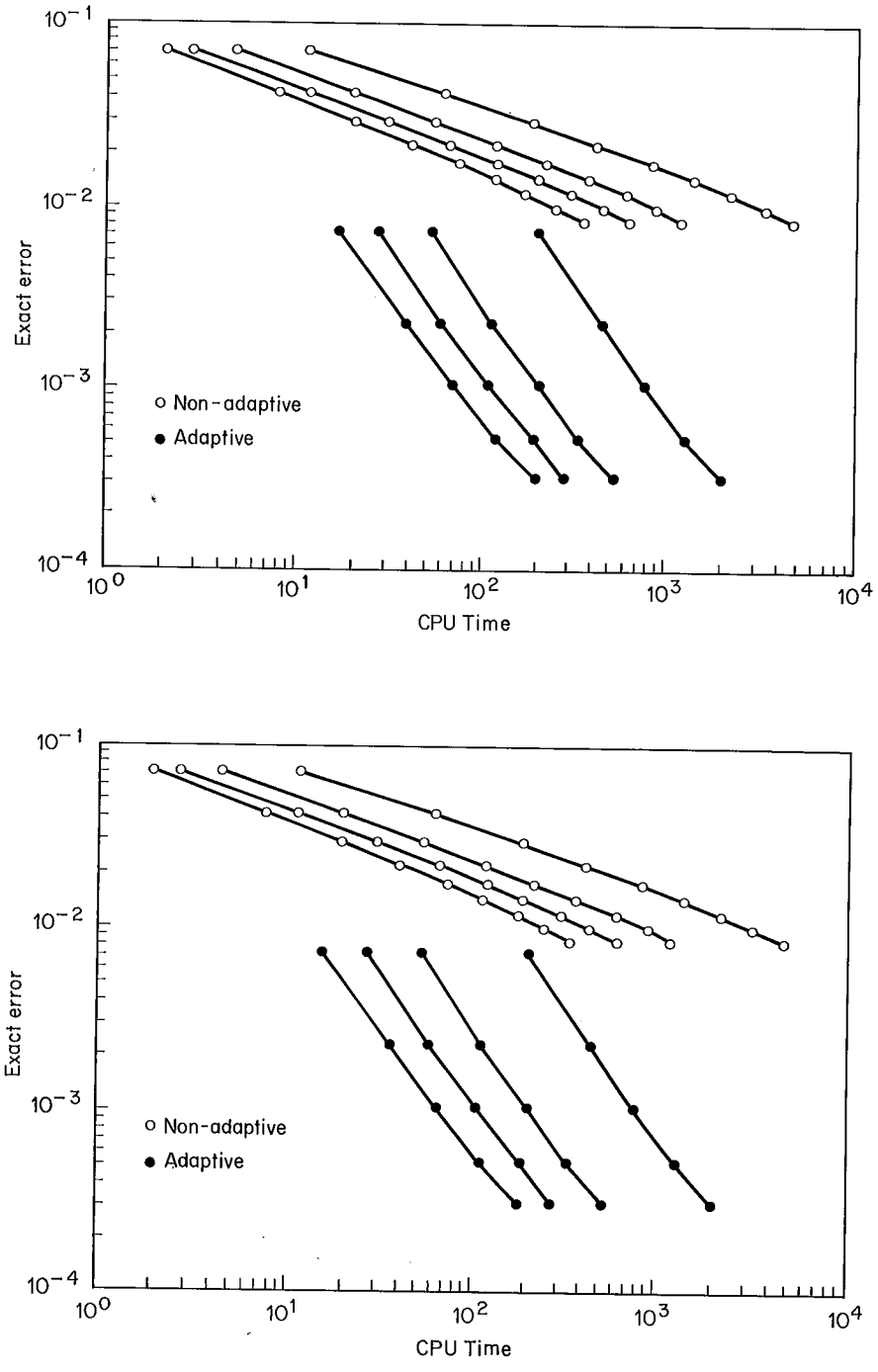

Figure 11. Global L^1 error as a function of CPU time for Example 3 using non-adaptive methods (×) and adaptive h-refinement methods (•) with static (top) and dynamic (bottom) load balancing. Computations are shown for systems using 1, 4, 8, and 15 processors (right to left for each set of curves).

The efficiency of this mesh refinement strategy and of the serial depth-first traversal and dynamic balancing techniques are appraised in an example. Performance of the parallel traversal procedure was not as good as either of these schemes and results are not presented for it. As in section 4, a 16-processor Sequent Balance 21000 computer was used for the experiment. CPU time and parallel speed up are used as performance measures.

Example 3. Consider the linear scalar differential equation

$$u_t + 2u_x + 2u_y = 0, \quad 0 < x, y < 1, \quad t > 0, \tag{5.7}$$

with initial and Dirichlet boundary data specified so that the exact solution is

$$u(x, y, t) = \frac{1}{2}\left[1 - \tanh(100x - 10y - 180t + 10)\right]. \tag{5.8}$$

The solution (5.8) is a relatively steep but smooth wave that moves at an angle of 45 degrees across the square domain as time progresses.

Adaptive refinement is controlled by using an approximation of the local discretization error in the L^1 norm as a refinement indicator. Exact errors for this scalar problem are also measured in L^1 as

$$||e(\cdot, \cdot, t)||_1 = \int\int_{\Omega} |\mathbf{P}u(x, y, t) - U(x, y, t)|\, dxdy, \tag{5.9}$$

where $U(x, y, t)$ is a piecewise constant representation of the discrete solution, $\mathbf{P}\, u\,(x, y, t)$ is a projection onto the space of piecewise constant functions obtained by using values at cell centers.

Our first experiment involves the solution of (5.7 & 5.8) for $0 < t \leq 0.35$ on 10×10, 25×25, and 45×45 uniform grids having initial time steps of 0.017, 0.007, and 0.004, respectively. No spatial refinement was performed and the static and dynamic load balancing strategies were used. CPU times and parallel speed-ups for each base mesh and each load balancing technique are shown in figure 9. Speed-up with 15 processors and the static load balancing technique (shown in the upper portion of figure 9) are in excess of 51, 75 and 87 percent of ideal with the 10×10, 25×25, and 45×45 base meshes, respectively. Speed-up increases dramatically with the finer meshes due to smaller data granularity. Similar speed-up data for the three base meshes with the dynamic load balancing technique (shown in the lower portion of figure 9) are 53, 77, and 90 percent of ideal. The static load balancing strategy takes slightly more time than the dynamic technique, except in the uni-processor case where they are identical, because of load imbalances on the P subdomains due to differences in the times required to generate initial and boundary data.

Our second experiment involves solving (5.7 & 5.8) for $0 < t \leq 0.35$ on a 10×10 base mesh having an initial time step of 0.017, using dynamic load balancing and adaptive h-refinement with either binary refinement or M-ary followed by 2-ary refinement. Refinement tolerances of 0.012, 0.006, and 0.003 were selected. The resulting CPU times and parallel speed-ups for each adaptive strategy are presented in figure 10. Maximum speed-ups shown in the upper portion of figure 10 for the binary refinement strategy are in excess of 82, 86, and 72 percent of ideal for tolerances of 0.012, 0.006, and 0.003, respectively. Initially, parallel performance improves as the tolerance is decreased due to

the finer data granularity; however, the performance ultimately degrades due to the serial overhead incurred when managing a more complex data structure. Maximum speed-ups for the more sophisticated M-ary followed by 2-ary refinement strategy shown in the lower portion of figure 10 are in excess of 88, 82, and 73 percent of ideal for the three decreasing tolerances. Speed-ups for this refinement strategy are only marginally better than those for the binary refinement technique, but the CPU times for the M-ary strategy are much less than those for the binary refinement strategy. For example, CPU times with 15 processors and a tolerance of 0.003 were 226.11 and 182.73 for the binary and M-ary strategies, respectively. Maintaining a shallow tree has clearly increased performance.

Speed-up is not an appropriate measure of the complexity required to solve a problem to a prescribed level of accuracy. Tradeoffs occur between the higher degree of parallelism possible with a uniform mesh solution and the greater sequential efficiency of an adaptive procedure. In order to gauge the differential, we generated uniform mesh and adaptive mesh solutions of (5.7 & 5.8) on various processor configurations and to varying levels of accuracy for both static serial tree traversal and dynamic load balancing strategies. Computations on uniform grids ranged from a 5×5 mesh to a 45×45 mesh. All adaptive computations used a 10×10 base mesh, M-ary followed by 2-ary refinement, and tolerances ranging from 0.012 to 0.003.

Results for the global L^1 error as a function of CPU time are presented in figure 11 for computations performed on 1, 4, 8, and 15 processor systems. Static and dynamic load balancing strategies are shown in the upper and lower portions of the figure, respectively. For each strategy, the upper set of curves, displaying non-adaptive results, are much less efficient and converging at a much slower rate than the adaptive solutions shown in the lower set of curves. The adaptive solutions are converging at a rate of approximately 1.4 relative to CPU time while the non-adaptive solutions are converging at a rate of approximately 0.4. These results demonstrate a strong preference for adaptive methods for all but the largest tolerances.

6. Future Investigations

The strategies and software described herein are far from being complete. Ongoing work with *pdefront* will provide symbolic interfaces to more of our software. Interfacing *pdefront* and the finite quadtree mesh generation system is a primary concern. With a substantial collection of algorithmic ideas and strategies for generating meshes, obtaining solutions, adaptive enrichment, and error estimation, we need a more effective way of organizing our software in order to determine optimal combinations of procedures for particular circumstances. The notion of a software laboratory as used by Kapur and Zhang (1988) appears to be suitable for our intentions.

High-order and hp-refinement strategies have the highest convergence rates on serial processors. Successful use of adaptive strategies in parallel environments depends heavily on the efficient implementation of these procedures on shared- and distributed-memory computers. The edge coloring procedure alluded to in section 4 should provide some improvement over existing strategies on shared-memory systems, but no procedure is available for using hp-refinement on data-parallel computers. High-order and hp-refinement

techniques are being added to our collection of methods for solving hyperbolic systems using the finite element methods (Cockburn and Shu, 1989). The p-hierarchical Legendre polynomial basis embedded in these methods should also furnish error estimates similar to those that we have developed for parabolic systems (Adjerid *et al.*, 1990). These techniques are far more efficient than Richardson's extrapolation.

Our h-refinement procedure for hyperbolic systems could be improved by beginning each base-mesh time step with an adaptively chosen mesh that utilizes known nonuniformities in the solution discovered during the previous base-mesh time step. Processors would still have to be scheduled to balance loads in this case, and procedures for doing this are unavailable. Finally, parallel procedures for distributed memory systems and procedures for three-dimensional problems are of great interest.

References

L. Adams and J. Ortega (1982), "A multi-color SOR method for parallel computation", K.E. Batcher, W.C. Meilander and J.L. Potter, eds., *Proc. Int. Conf. Parallel Processing*, Comput. Soc. Press, Silver Spring, 53–56.

S. Adjerid and J.E. Flaherty (1986), "A moving mesh finite element method with local refinement for parabolic partial differential equations", *Comput. Meths. Appl. Mech. Eng.*, **56**, 3–26.

S. Adjerid, J.E. Flaherty and Y. Wang (1990), "A posteriori error estimation with finite element methods of lines for one-dimensional parabolic systems", in preparation.

D.C. Arney, R. Biswas and J.E. Flaherty (1990), *An Adaptive Mesh Moving and Refinement Procedure for One-Dimensional Conservation Laws,* Technical Report 90-6, Dept. of Comput. Sci., Rensselaer Polytechnic Institute, Troy, NY.

D.C. Arney and J.E. Flaherty (1986), "A two-dimensional mesh moving technique for time-dependent partial differential equations", *J. Comput. Phys.*, **67**, 124–144.

D.C. Arney and J.E. Flaherty (1990), "An adaptive mesh-moving and local refinement method for time-dependent partial differential equations", *ACM Trans. Math. Software*, **16**, 48–71.

U.M. Ascher, R.M. Mattheij and R.D. Russell (1988), *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, NJ.

O. Axelsson and V.A. Barker (1984), *Finite Element Solution of Boundary Value Problems: Theory and Computation*, Academic Press, Orlando, FL.

I. Babuska and E. Rank (1986), *An Expert-system Like Approach in the Hp-Version of the Finite Element Method,* Technical Note BN-1084, Institute for Physical Sci. and Tech., University of Maryland, College Park.

I. Babuska, B.A. Szabo and I.N. Katz (1981), "The p-version of the finite element method", *SIAM J. Numer. Anal.*, **18**, 515–545.

I. Babuska and D. Yu (1990), "A posteriori error estimation for biquadratic elements and adaptive approaches", in preparation.

P.L. Baehmann, S.L. Wittchen, M.S. Shephard, K.R. Grice and M.A. Yerry (1987), "Robust, geometrically based, automatic two-dimensional mesh generation", *Int. J. Numer. Meths. Eng.*, **24**, 1043–1078.

R.E. Bank (1990), *PLTMG: A Software Package for Solving Elliptic Partial Differential Equations. Users' Guide 6.0*, Frontiers in Appl. Math. 7, SIAM, Philadelphia.

M. Benantar, R. Biswas, J.E. Flaherty and M.S. Shephard (1990), "Parallel computation with adaptive methods for elliptic and hyperbolic systems", *Comput. Meths. Appl. Mech. Eng.*, **82**, 73–93.

B. Cockburn and C.-W. Shu (1989), "TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws II: general framework", *Math. Comput.*, **52**, 411–435.

J.E. Flaherty and M. Benantar (1990), "Parallel element-by-element techniques for elliptic systems using finite quadtree meshes", in preparation.

J.E. Flaherty, P.J. Paslow, M.S. Shephard and J.D. Vasilakis (1989), *Adaptive Methods for Partial Differential Equations*, SIAM, Philadelphia.

C.W. Gear (1971), *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, NJ.

I. Gustafsson and G. Lindskog (1986), "A preconditioning technique based on element matrix factorizations", *Comput. Meths. Appl. Mech. Eng.*, **55**, 201–220.

D. Kapur and H. Zhang (1988), "RRL: a rewrite rule laboratory", *Lecture Notes in Comput. Sci. 310,*

E. Lusk and R. Overbeek, eds., Springer-Verlag, Berlin, 768–769.

R.B. King and V. Sonnad (1987), "Implementation of an element-by-element solution algorithm for the finite element method on a coarse-grained parallel computer", *Comput. Meths. Appl. Mech. Eng.*, **65**, 47–59.

P.K. Moore and J.E. Flaherty (1990), *Adaptive Local Overlapping Grid Methods for Parabolic Systems in Two Space Dimensions,* Technical Report 90-5, Dept. of Comput. Sci., Rensselaer Polytechnic Institute, Troy, NY.

B. Nour-Omid and B.N. Parlett (1985), "Element preconditioning using splitting techniques", *SIAM J. Sci. Stat. Comput.*, **6**, 761–770.

J.M. Ortega (1988), *Introduction to Parallel and Vector Solution of Linear Systems*, Plenum Press, NY.

R.D. Richtmyer and K.W. Morton (1967), *Difference Methods for Initial-Value Problems*, Interscience, NY.

S. Steinberg and P.J. Roache (1985), *Applications of Computer Algebra*, R. Pavelle, ed., Kluwer Academic Publisher, Boston, MA, 74–93.

S. Steinberg and P.J. Roache (1986), "Using MACSYMA to write FORTRAN subroutines", *J. Symbolic Computation,* **2**, 213–216.

S. Steinberg and P.J. Roache (1988a), "A toolkit of symbolic manipulation programs for variational grid generation", *Coupling Symbolic and Numerical Computing in Expert Systems, II,* North-Holland, NY, 103–116.

S. Steinberg and P.J. Roache (1988b), "Automatic generation of finite-difference code", *Proc. Symbolic Computation in Fluid Mechanics and Heat Transfer,* H.H. Bau, T. Herbert and M.M. Yovanovich, eds., HTD-105, AMD-97, Chicago, IL.

B. Szabo and I. Babuska (1990), *Introduction to Finite Element Analysis,* John Wiley & Sons, to appear.