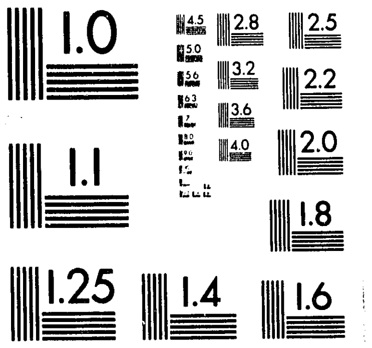

1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

A Massively Parallel Adaptive Finite Element Method with Dynamic Load Balancing

Karen D. Devine[1,2] and Joseph E. Flaherty[1]
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, NY 12180
kddevin@cs.sandia.gov
flaherje@cs.rpi.edu

Stephen R. Wheat[2]
MP Computing Research Laboratory
Department 1424
Sandia National Laboratories
Albuquerque, NM 87185-5800
srwheat@cs.sandia.gov

Arthur B. Maccabe[2,3]
Department of Computer Science
The University of New Mexico
Albuquerque, NM 87131
maccabe@cs.unm.edu

Abstract: We construct massively parallel, adaptive finite element methods for the solution of hyperbolic conservation laws in one and two dimensions. Spatial discretization is performed by a discontinuous Galerkin finite element method using a basis of piecewise Legendre polynomials. Temporal discretization utilizes a Runge-Kutta method. Dissipative fluxes and projection limiting prevent oscillations near solution discontinuities. The resulting method is of high order and may be parallelized efficiently on MIMD computers. We demonstrate parallel efficiency through computations on a 1024-processor nCUBE/2 hypercube. We also present results using adaptive p-refinement to reduce the computational cost of the method. We describe tiling, a dynamic, element-based data migration system. Tiling dynamically maintains global load balance in the adaptive method by overlapping neighborhoods of processors, where each neighborhood performs local load balancing. We demonstrate the effectiveness of the dynamic load balancing with adaptive p-refinement examples.

1. Introduction

We are studying massively parallel, adaptive finite element methods for the solution of systems of d-dimensional hyperbolic conservation laws of the form

$$\mathbf{u}_t + \sum_{i=1}^{d} \mathbf{f}_i(\mathbf{u})_{x_i} = 0, \ \mathbf{x} \in \Omega, \ t > 0, \tag{1a}$$

subject to the initial conditions

$$\mathbf{u}(\mathbf{x}, 0) = \mathbf{u}^0(\mathbf{x}), \ \mathbf{x} \in \Omega \cup \partial\Omega, \tag{1b}$$

and appropriate well-posed boundary conditions on $\partial\Omega$. High-order methods and the combination of mesh refinement and order variation (hp-refinement) have been shown to produce effective solution techniques for elliptic [16] and parabolic [1, 2, 3] problems. It is, thus, natural to determine whether or not they will be as efficient when applied to hyperbolic systems. To achieve high-order accuracy, finite difference schemes for (1), such as Total Variation Diminishing (TVD) [21, 23] and Essentially Non-Oscillatory (ENO) [19] schemes, use a computational stencil that enlarges with order. The wide stencil makes the methods difficult to implement near irregular boundaries, and limits efficient implementation on massively parallel computers, since data may have to be passed between non-adjacent processors. Finite element methods,

1 This work was partially supported by Sandia National Laboratories under Research Agreement #67-8709.
2 This work was performed at Sandia National Laboratories, operated for the U.S. Department of Energy under contract #DE-AC04-76DP00789.
3 On Faculty Sabbatical to Sandia National Laboratories.

however, can easily model problems having complicated geometries and have stencils that are invariant with method order.

With a motivation to explore adaptive high-order parallel methods, we abandon the traditional finite element formulation in favor of a local discontinuous Galerkin method of Cockburn and Shu [7, 8, 9, 18]. Spatially, the solution is approximated by a basis of piecewise Legendre polynomials that are continuous on an element, but may have discontinuities at interelement boundaries. By not enforcing global continuity, we hope to approximate discontinuous solutions of (1) more accurately. Fluxes at element boundaries are computed by solving an approximate Riemann problem with a projection limiter applied to keep the average solution monotone near discontinuities [23]. An adaptive limiting procedure maintains high-order accuracy near smooth extrema while improving global monotonicity near discontinuities relative to other techniques [9]. Time discretization is performed by an explicit Runge-Kutta method.

Parallel performance in one and two dimensions has nearly perfect scaled speed-up [10] when the local Galerkin method is implemented on an NCUBE/2 hypercube computer. When adaptive p-refinement is incorporated into the method, parallel performance substantially degrades due to processor load imbalance. To determine an efficient domain decomposition and processor mapping, many massively parallel finite element methods use *static* load balancing [11, 12, 13] as a preprocessor to the finite element calculation. Adaptive methods, however, require *dynamic* load balancing to adjust changing processor loads as the computation proceeds. We have developed a dynamic, fine-grained, element-based data migration algorithm called *tiling* that maintains global load balance by overlapping neighborhoods of processors, where each neighborhood performs local load balancing. The tiling system supports a large class of finite element and finite difference applications. It provides an automatic element management system library to which a programmer integrates the application by providing subroutines for the application's data exchange pattern, element processing, and boundary processing. Dynamic migration of elements from processor to processor is automatically performed by the run-time system.

The effectiveness of the tiling system has been demonstrated for non-adaptive finite difference and finite element methods whose geometries and boundary conditions create load imbalance [24]. We incorporate the adaptive p-refinement method into the tiling system to recover the parallel efficiency lost to load imbalance in the adaptive method, and demonstrate the tiling system's effectiveness on problems with changing work loads with several experiments on the nCUBE/2.

2. The Discontinuous Galerkin Method in One Dimension

To simplify the presentation, consider a one-dimensional ($d = 1$) system of conservation laws (1), and partition Ω into subintervals (x_{j-1}, x_j), $j = 1, 2, \ldots, J$. Construct a weak form of the problem by multiplying (1a) by a test function $\mathbf{v} \in L^2(x_{j-1}, x_j)$ and integrating the result on (x_{j-1}, x_j) while integrating the flux term by parts to obtain

$$\frac{d}{dt}\int_{x_{j-1}}^{x_j} \mathbf{v}^T\mathbf{u}\,dx + \mathbf{v}^T\mathbf{f}(\mathbf{u})\Big|_{x_{j-1}}^{x_j} - \int_{x_{j-1}}^{x_j} \mathbf{v}_x^T\mathbf{f}(\mathbf{u})\,dx = 0, \text{ for all } \mathbf{v} \in L^2(x_{j-1}, x_j). \tag{2}$$

Use a linear transformation to map (x_{j-1}, x_j) onto a "canonical element" $-1 \le \xi \le 1$ and obtain

$$\frac{\Delta x_j}{2}\frac{d}{dt}\int_{-1}^{1} \mathbf{v}^T\mathbf{u}\,d\xi + \mathbf{v}^T\mathbf{f}(\mathbf{u})\Big|_{-1}^{1} - \int_{-1}^{1} \mathbf{v}_\xi^T\mathbf{f}(\mathbf{u})\,d\xi = 0, \text{ for all } \mathbf{v} \in L^2(-1, 1), \tag{3a}$$

where

$$\Delta x_j = x_j - x_{j-1}, j = 1, 2, \ldots, J. \tag{3b}$$

Approximate $\mathbf{u}(\xi, t) \in L^2(-1, 1)$ by the p^{th} degree polynomial $\mathbf{U}_j(\xi, t)$ expressed in terms of a basis of Legendre polynomials as

$$\mathbf{u}(\xi, t) \approx \mathbf{U}_j(\xi, t) = \sum_{k=0}^{p} \mathbf{c}_{jk}(t)P_k(\xi), \xi \in (-1, 1). \tag{4}$$

Substituting the polynomial approximation (4) into (3), selecting $\mathbf{v}$ to be proportional to $P_k(\xi)$, and using the orthogonality properties of Legendre polynomials [22], we determine $\mathbf{c}_{jk}$, $j = 1, 2, \ldots, J$, $k = 0, 1, \ldots, p$,

$$\frac{\Delta x_j}{2k+1}\dot{\mathbf{c}}_{jk} + \mathbf{f}(\mathbf{U}_j(1, t)) - (-1)^k\mathbf{f}(\mathbf{U}_j(-1, t)) - \int_{-1}^{1} P_k'(\xi)\mathbf{f}(\mathbf{U}_j)d\xi = 0. \tag{5a}$$

The $\mathbf{c}_{jk}$ are initialized by L^2 projection of the initial data (1) onto the space of Legendre polynomials, i.e.,

$$\int_{-1}^{1} P_k(\xi)\,[\mathbf{U}_j(\xi, 0) - \mathbf{u}^0(x(\xi))]\,d\xi = 0, j = 1, 2, \ldots, J, k = 0, 1, \ldots, p. \tag{5b}$$

Integral terms in (5a) are evaluated exactly for linear problems, using the properties of Legendre polynomials [22], or numerically using $(K+1)/2$-point Gauss-Legendre quadrature. The boundary flux $\mathbf{f}(\mathbf{U}_j(1,t))$ is approximated by a numerical flux function $\mathbf{h}(\mathbf{U}_j(1,t), \mathbf{U}_{j+1}(-1,t))$. Presently, we use a Lax-Friedrichs numerical flux

$$\mathbf{h}(\mathbf{U}_L, \mathbf{U}_R) = \frac{1}{2}[\mathbf{f}(\mathbf{U}_L) + \mathbf{f}(\mathbf{U}_R) - |\lambda|(\mathbf{U}_R - \mathbf{U}_L)], \tag{5c}$$

where λ is the maximum eigenvalue of the Jacobian $\mathbf{f}_{\mathbf{u}}(\mathbf{U})$ on $\mathbf{U}_L \le \mathbf{U} \le \mathbf{U}_R$. Other numerical flux functions [9, 19] may improve performance and we shall investigate this possibility. Runge-Kutta integration of order p is used for temporal integration.

In regions where the solution of (1) is smooth, the scheme (4, 5) produces the $O(\Delta x^{p+1})$,

$$\Delta x = \max_{j=1,2,\ldots,J} \Delta x_j, \tag{6}$$

convergence expected for a p^{th}-degree approximation [9]. When $p > 0$, projection limiting is needed to prevent spurious oscillations near solution discontinuities. With projection limiting, the solution $\mathbf{U}_j(\xi, t)$, $j = 1, 2, \ldots, J$, is restricted after each Runge-Kutta stage to eliminate oscillations. Cockburn and Shu [9] describe a procedure for the projection limiting of scalar problems that prevents the approximate solution on an element from taking values outside of the range spanned by the neighboring solution averages by comparing deviations at element endpoints with differences of neighboring elements' average values. While preserving monotonicity of the average numerical solution, this limiting scheme flattens solutions near smooth extrema so that first-order accuracy is obtained there. To overcome this deficiency, we developed a limiting scheme that maintains monotonicity of solution moments on neighboring elements. (For comparisons of the two limiting schemes, see [6].) Using orthogonality properties of Legendre polynomials and (4), solution moments of a scalar problem are given by

$$\int_{-1}^{1} U_j P_k(\xi)\,d\xi = \frac{2}{2k+1} c_{jk}, \quad k = 0, 1, \ldots, p-1. \tag{7a}$$

Thus, to keep the k^{th} moment monotone, we must keep c_{jk} monotone on neighboring elements. Differentiating (4),

$$\frac{\partial^k}{\partial \xi^k} U_j(\xi, t) = \prod_{m=1}^{k} (2m-1) c_{jk} + \prod_{m=1}^{k+1} (2m-1) c_{j,k+1} \xi + \sum_{m=k+2}^{p} c_{jm}(t) \frac{d^k}{d\xi^k} P_m(\xi). \tag{7b}$$

Then, to keep c_{jk}, $k = 0, 1, \ldots, p-1$, monotone, we limit $c_{j,k+1}$ as

$$(2k+1)\, c_{j,k+1} = minmod((2k+1)\, c_{j,k+1}, c_{j+1,k} - c_{j,k}, c_{j,k} - c_{j-1,k}). \tag{7c}$$

where

$$minmod(a, b, c) = \begin{cases} \mathrm{sgn}(a) \min(|a|, |b|, |c|), & \text{if} \quad \mathrm{sgn}(a) = \mathrm{sgn}(b) = \mathrm{sgn}(c) \\ 0, & \text{otherwise.} \end{cases} \tag{7d}$$

The limiter (7) is applied adaptively. First, the highest-order coefficient is limited. Then the limiter is applied to successively lower-order coefficients when the next higher coefficient on the interval has been changed by the limiting. In this way, the limiting is applied only where it is needed, and accuracy is retained in smooth regions.

For vector systems, the scalar limiting function can be applied component-wise; however, Cockburn, et al. [8] showed that this simple extension of the limiting does not have a Total Variational Bounded (TVB) theory even for linear systems. Indeed, they observed small oscillations in their computational examples. To improve accuracy at the price of additional computation, we apply the limiter to the characteristic fields of the system [8, 14]. The diagonalizing matrices $\mathbf{T}(\mathbf{u})$ and $\mathbf{T}^{-1}(\mathbf{u})$ (consisting of the right and left eigenvectors of the Jacobian $\mathbf{f}_{\mathbf{u}}$) are evaluated using the average values of $\mathbf{U}_j$, $j = 1, 2, \ldots, J$. The scalar projection limiter is applied to each field of the characteristic vector. The result is then projected back to the physical space by post-multiplication by $\mathbf{T}^{-1}(\mathbf{U}_j)$.

Example 1. Consider the linear scalar problem

$$u_t + u_x = 0, -\pi < x < \pi, t > 0, \tag{8a}$$

$$u(x, 0) = \sin(x), -\pi \le x \le \pi, \tag{8b}$$

with periodic boundary data.

We verify the convergence rate of the one-dimensional method by solving (8) using (5) with no limiting. In Figure 1, we show the global L^1-error

$$\|u(\cdot, t) - U(\cdot, t)\|_1 = \int_{-L}^{L} |u(x, t) - U(x, t)|\, dx \tag{9}$$

versus the number of elements for $p = 0, 1, \ldots, 4$. The slope of the lines indicates that the order of the method is $O(\Delta x^{p+1})$.

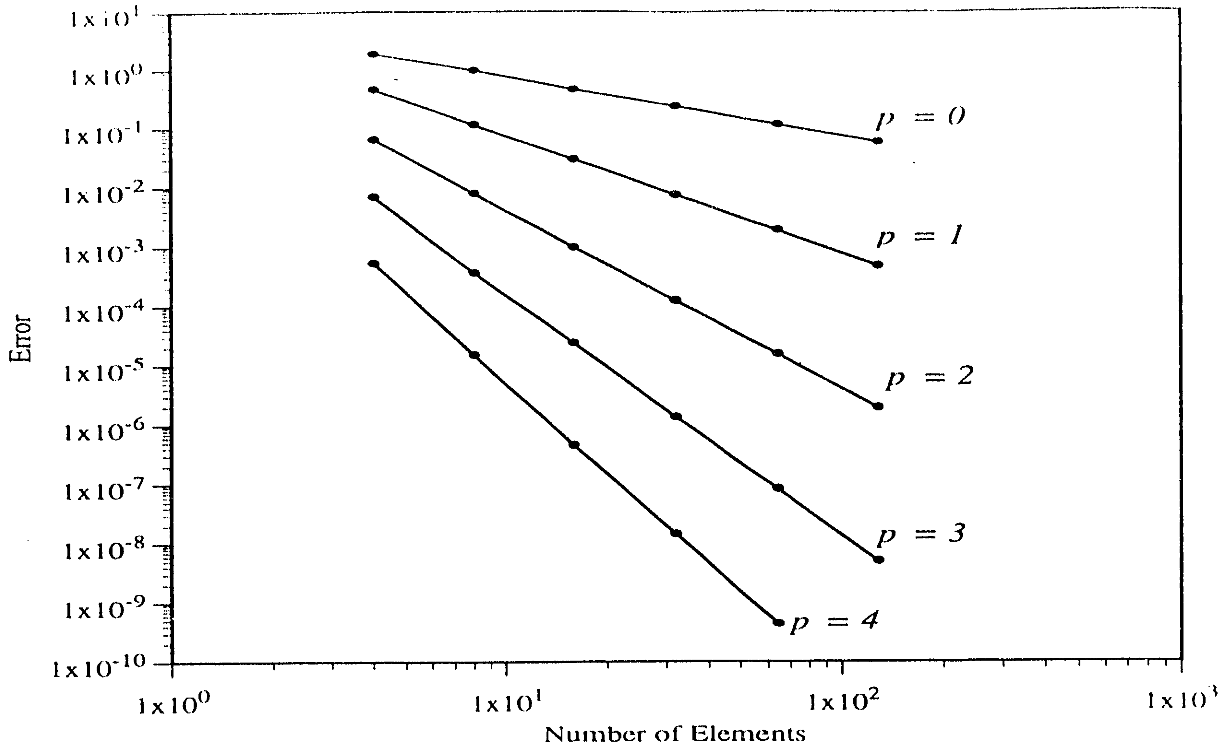


Figure 1. *L^1-error vs. the number of elements for Example 1 with $p = 0$ to 4.*

Example 2. Consider the periodic initial value problem for Burgers' equation

$$u_t + \left(\frac{u^2}{2}\right)_x = 0, \; t > 0, \tag{10a}$$

$$u^0(x) = \frac{1}{2} + \frac{1}{2}\sin(\pi x). \tag{10b}$$

Solutions of (10) at time $t = 1.1$, obtained on a 32-element mesh for $p = 0$, 1, and 2 using an upwind numerical flux

$$h(U_L, U_R) = \begin{cases} f(U_L), & \text{if } f'(u) \geq 0 \\ f(U_R), & \text{if } f'(u) < 0 \end{cases}$$

with moment limiting (7), are shown in Figure 2. The improved solution accuracy when p is increased from 0 to 2 can easily be seen. With $p = 2$, the limiter (7) maintained third-order accuracy in smooth regions of the solution (see Table 1), produced a sharp shock, and preserved average as well as global monotonicity on all but one subinterval.

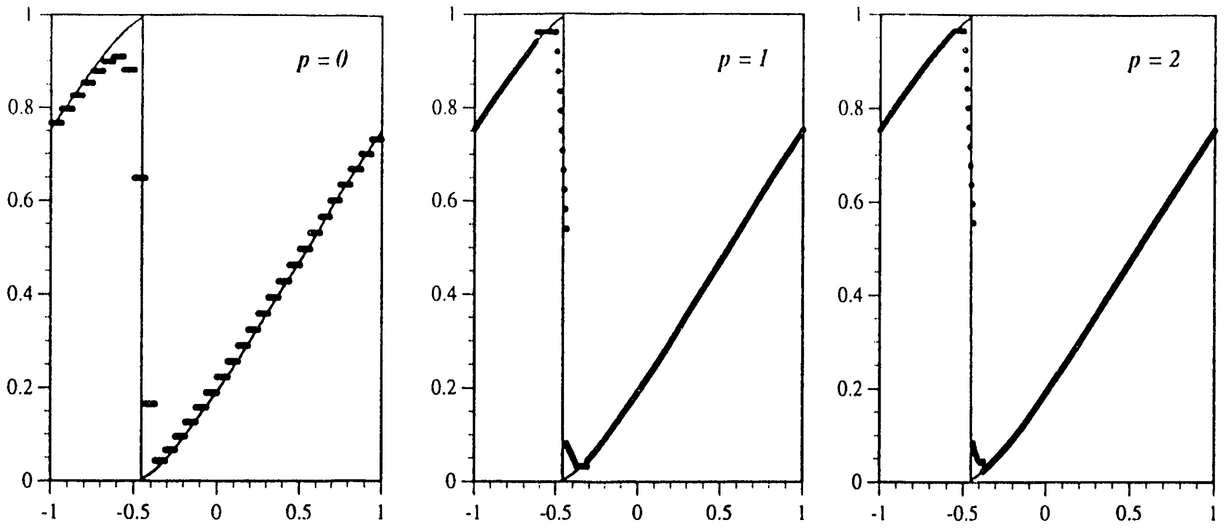


Figure 2. Exact (line) and numerical (•) solutions of Example 2 for p=0, 1, and 2 using the moment limiter (7). The improved solution accuracy for p=2 is clearly seen. (Finite element solutions are shown at eleven points per element.)

Number of Elements	Error in L^1 Norm	Order
32	2.39585e-05	
64	1.64509e-06	3.86
128	1.68787e-07	3.28
256	1.79387e-08	3.23
512	1.90090e-09	3.23

Table 1. Convergence of (5) with $p = 2$ and limiter (7) in smooth regions of Example 2. Error was measured in smooth region $x \in (-1, -0.5675) \cup (-0.375, 1)$ using (9).

Example 3. The one-dimensional Euler equations of gas dynamics can be written in the form (1), with $\mathbf{u}(x, t) = [\rho, m, e]^T$, and $\mathbf{f}(\mathbf{u}(x, t)) = [\rho q, mq + P, eq + Pq]^T$, where ρ, P, m, q, and e are the density, pressure, momentum ($m = \rho q$), velocity, and energy, respectively. The system is completed with the ideal equation of state

$$P = (\gamma - 1)\left(e - \frac{1}{2}\rho q^2\right)$$

where γ is the ratio of specific heats, taken here as 1.4. We consider Sod's shock tube Riemann problem [20]

$$[\rho^0, q^0, P^0]^T = \begin{cases} [1, 0, 1]^T, & \text{if } x \le 0.5 \\ [0.125, 0, 0.1]^T, & \text{if } x > 0.5 \end{cases}$$

The problem is solved using a piecewise quadratic approximation ($p = 2$) on a 64-element mesh. In Figure 3, we show the density, pressure, and velocity at time $t = 0.1$ using limiter (7). The solution method sharply captures both shocks and contact discontinuities, and the high-order coefficients are determined to preserve average and, to a large extent, global monotonicity.

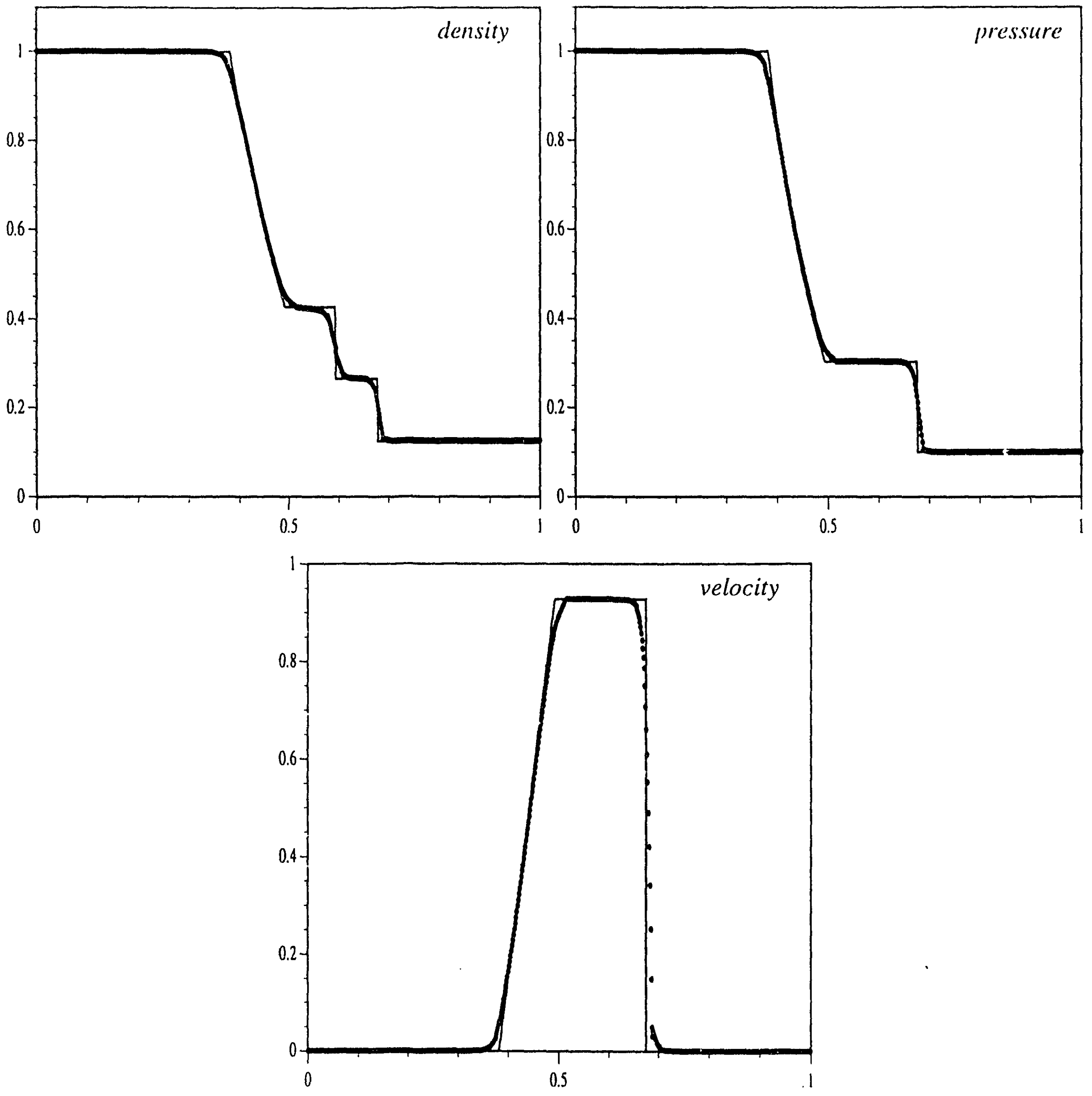


Figure 3. Density, pressure, and velocity at t=0.1 for Example 3 with p =2 using the moment limiter. (Solutions are shown at eleven points per element.)

3. The Discontinuous Galerkin Method in Two Dimensions

The two-dimensional method is a direct extension of the one-dimensional method. For simplicity, let us write (1) as

$$\mathbf{u}_t + \mathbf{f}(\mathbf{u})_x + \mathbf{g}(\mathbf{u})_y = 0, \ (x, y) \in \Omega, \ t > 0, \tag{11a}$$

$$\mathbf{u}(x, y, 0) = \mathbf{u}^0(x, y), \ (x, y) \in \Omega \cup \partial\Omega. \tag{11b}$$

Restricting Ω to be rectangular, partition it into rectangular elements

$$\Omega_{ij} = \{ (x, y) | x_{i-1} \le x \le x_i, y_{j-1} \le y \le y_j \}, \ i = 1, 2, \ldots, I, \ j = 1, 2, \ldots, J.$$

Representing $\mathbf{u}(x, y, t)$ on Ω_{ij} by a basis of tensor products of Legendre polynomials on the canonical element $\Omega_C = \{ (\xi, \eta) | -1 \le \xi, \eta \le 1 \}$

$$\mathbf{u}(\xi, \eta, t) \approx \mathbf{U}_{ij}(\xi, \eta, t) = \sum_{k=0}^{p} \sum_{m=0}^{p} \mathbf{c}_{ijkm} P_k(\xi) P_m(\eta), \ (\xi, \eta) \in \Omega_C, \tag{12}$$

and constructing a weak form of (11) as in Section 2, we find

$$\dot{\mathbf{c}}_{ijkm} = -\frac{(2k+1)(2m+1)}{2\Delta x_i \Delta y_j} (\mathbf{I}_1 + \mathbf{I}_2 + \mathbf{I}_3), \ k, m = 0, 1, \ldots, p, \tag{13a}$$

where

$$\mathbf{I}_1 = -\int_{-1}^{1}\int_{-1}^{1} [\Delta y_j P_k'(\xi) P_m(\eta) \mathbf{f}(\mathbf{U}_{ij}) + \Delta x_i P_k(\xi) P_m'(\eta) \mathbf{g}(\mathbf{U}_{ij})] \, d\xi d\eta, \tag{13b}$$

$$\mathbf{I}_2 = \Delta y_j \int_{-1}^{1} [P_m(\eta) \mathbf{f}(\mathbf{U}_{ij}(1, \eta, t)) - (-1)^k P_m(\eta) \mathbf{f}(\mathbf{U}_{ij}(-1, \eta, t))] \, d\eta, \tag{13c}$$

$$\mathbf{I}_3 = \Delta x_i \int_{-1}^{1} [P_k(\xi) \mathbf{g}(\mathbf{U}_{ij}(\xi, 1, t)) - (-1)^m P_k(\xi) \mathbf{g}(\mathbf{U}_{ij}(\xi, -1, t))] \, d\xi. \tag{13d}$$

The boundary fluxes $\mathbf{f}(\mathbf{U}_{ij}(\pm 1, \eta, t))$ and $\mathbf{g}(\mathbf{U}_{ij}(\xi, \pm 1, t))$, and initial data $\mathbf{c}_{ijkm}(0)$ are approximated by the Lax-Friedrichs numerical flux (5), and L^2-projection respectively.

Following Biswas [5], we apply the one-dimensional projection limiter (7) along each of the two spatial directions. Thus, for a p^{th} degree approximation ($p \ge 1$), we first limit the coefficients c_{ijp0} and c_{ij0p}. If further limiting is necessary and if $p \ge 2$, we also limit the coefficients $c_{ij,\,p-1,\,0}$ and $c_{ij,\,0,\,p-1}$, and continue as in the one-dimensional case. "Cross-product" coefficients, c_{ijpp}, c_{ijkp}, and c_{ijpk}, $k \ne p \ge 1$, are not limited. Biswas [5] conjectured that these coefficients have a lesser effect on the numerical solution than either c_{ijp0} or c_{ij0p} since they are at least one order higher. Our experimental results and those of Biswas [5] indicate this to be true; however, a more rigorous analysis is necessary.

4. Adaptive p-Refinement

We have begun experiments with an adaptive p-refinement version of the two-dimensional method (13) using a method-of-lines approach. A spatial error estimate is used to control order variation procedures that attempt to keep the global L^1-error (9) less than a specified tolerance TOL by maintaining

$$E_{ij}(t) \le \mathrm{TOL}_L = \frac{\mathrm{TOL}}{IJ}, \; i = 1, 2, \ldots, I, \; j = 1, 2, \ldots, J, \tag{14}$$

where E_{ij} is the maximum local L^1-error estimate of the solution vector on element Ω_{ij}. For these initial experiments, we use a p-refinement spatial error estimate:

$$E_{ij}(t) = \left\| \int_{-1}^{1}\int_{-1}^{1} \left| \mathbf{U}_{ij}^{p+1}(\xi, \eta, t) - \mathbf{U}_{ij}^{p}(\xi, \eta, t) \right| d\xi d\eta \right\|_{\infty} \tag{15}$$

where $\mathbf{U}_{ij}^{p}$ is the p^{th}-degree approximation of $\mathbf{u}$. While this estimate is computationally expensive, it is still less expensive than mesh-refinement estimates such as Richardson's extrapolation [17] and can be used to reduce the effort involved in recomputing $\mathbf{U}_{ij}$ and its error estimate when enrichment is needed. Instead of recomputing $\mathbf{U}_{ij}(t+\Delta t)$ when a higher-order approximation is needed, set $\mathbf{U}_{ij}(t+\Delta t) = \mathbf{U}_{ij}^{p+1}(t+\Delta t)$. Initialize the new error estimate at time t

$$\mathbf{U}_{ij}^{p+2}(t) = \mathbf{U}_{ij}^{p+1}(t) + \sum_{m=0}^{p+2} [\mathbf{c}_{ij,p+2,m} P_{p+2}(\xi) P_m(\eta) + \mathbf{c}_{ij,m,p+2} P_m(\xi) P_{p+2}(\eta)] \tag{16}$$

with $\mathbf{c}_{ij,p+2,m} = \mathbf{c}_{ij,m,p+2} = 0$, $m = 0, 1, \ldots, p+2$, and recompute only $\mathbf{U}_{ij}^{p+2}$ over the time step.

We initialize $\mathbf{U}_{ij}$, $i = 1, 2, \ldots, I$, $j = 1, 2, \ldots, J$, to the lowest-degree polynomial satisfying

$$\left\| \int_{x_{i-1}}^{x_i}\int_{y_{j-1}}^{y_j} \left| \mathbf{u}^0(x, y) - \mathbf{U}_{ij}(\xi(x), \eta(y), 0) \right| dydx \right\|_{\infty} \le \mathrm{TOL}_L. \tag{17}$$

After each time step, we compute E_{ij}, $i = 1, 2, \ldots, I$, $j = 1, 2, \ldots, J$. If $E_{ij} > \mathrm{TOL}_L$, we increase the polynomial degree of $\mathbf{U}_{ij}$. The solution $\mathbf{U}_{ij}$ and the error estimate are recomputed on enriched elements until $E_{ij} \le \mathrm{TOL}_L$ on all elements when the time step is accepted.

Additional computational savings are possible by predicting the degree of the approximation needed to satisfy the accuracy requirements during the next time step. After a time step is accepted, if $E_{ij} > H_{max}\mathrm{TOL}_L$ for $H_{max} \in (0, 1]$, increase the degree of $\mathrm{U}_{ij}(t+\Delta t)$ by setting $\mathrm{U}_{ij}(t+\Delta t) = \mathrm{U}_{ij}^{p+1}(t+\Delta t)$, and define $\mathrm{U}_{ij}^{p+2}(t+\Delta t)$ according to (16). If $E_{ij} < H_{min}\mathrm{TOL}_L$ for $H_{min} \in [0, 1)$, decrease the degree of $\mathbf{U}_{ij}(t+\Delta t)$ by setting $\mathbf{c}_{ijpm} = \mathbf{c}_{ijmp} = 0$, $m = 0, 1, \ldots, p$, and of $\mathrm{U}_{ij}^{p+1}(t+\Delta t)$ by setting $\mathbf{c}_{ij,\,p+1,\,m} = \mathbf{c}_{ij,\,m,\,p+1} = 0$, $m = 0, 1, \ldots, p+1$.

Example 4. We solve

$$u_t + 2u_x + 2u_y = 0,\ 0 \le x, y \le 1,\ t > 0, \tag{18a}$$

by both fixed-order and adaptive-order methods on $0 < t \le 0.1$ with initial and Dirichlet boundary conditions specified so that the exact solution is

$$u(x, y, t) = \frac{1}{2}(1 - \tanh(20x - 10y - 20t + 5)),\ 0 \le x, y \le 1. \tag{18b}$$

In Figure 4, we show the exact solution of (18) at time $t = 0$ and the adaptive 16×16-element mesh generated to satisfy the initial data for $TOL_L = 1.0 \times 10^{-5}$.

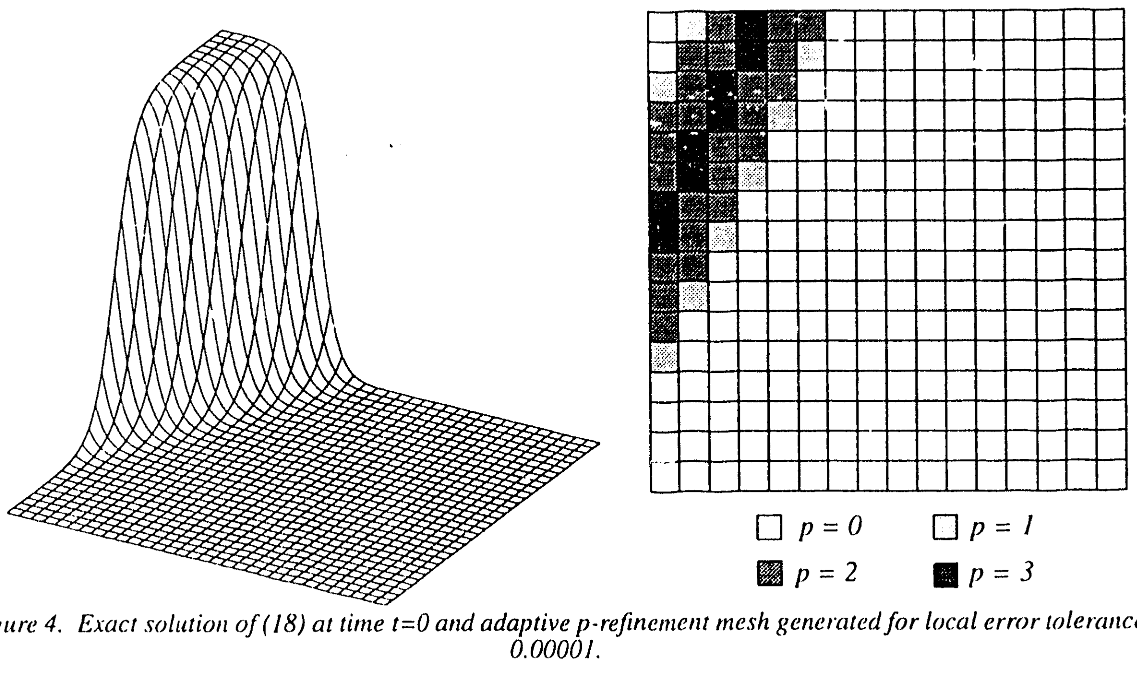


Figure 4. Exact solution of (18) at time t=0 and adaptive p-refinement mesh generated for local error tolerance 0.00001.

In Figure 5, we show the global L^1-error versus the CPU time for the fixed-order method with $p = 0$, 1, and 2 on 8×8, 16×16, 32×32, and 64×64-element meshes, and the p-adaptive method, $H_{max} = 0.9$, $H_{min} = 0.1$, and local error tolerances TOL_L ranging from 5×10^{-9} to 5×10^{-4} on a

16 × 16-element mesh. The p-adaptive method requires more computation than the fixed-order methods for large error tolerances, but because of its faster convergence rate, it requires less work than the fixed-order methods to obtain small errors.

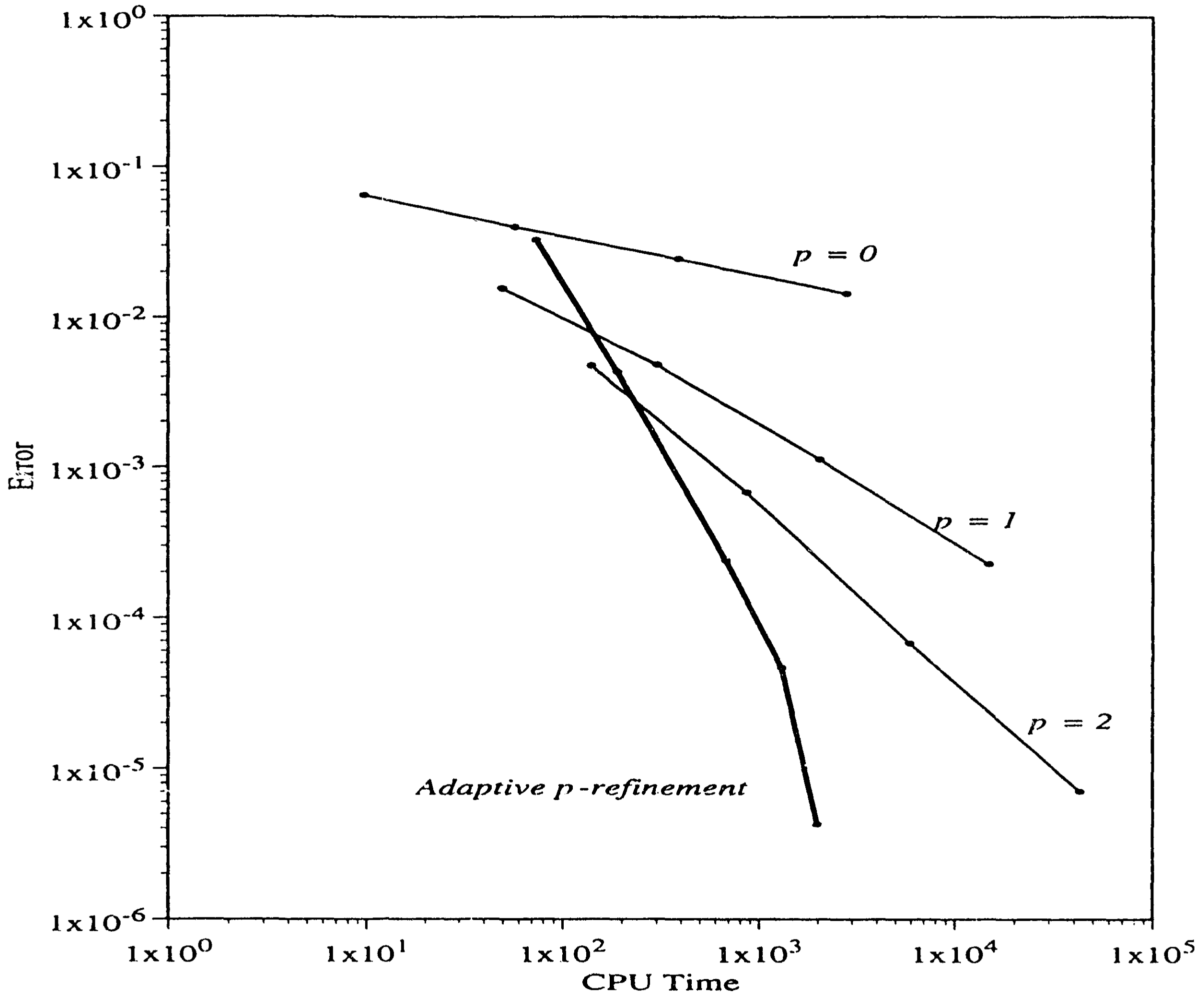


Figure 5. Convergence of the p-adaptive method compared with fixed-order methods for Example 4.

5. Parallel Implementation of the One- and Two-Dimensional Methods

Both the one- and two-dimensional methods are well suited to parallelization on massively parallel computers. The computational stencil involves only nearest-neighbor communication regardless of the degree of the piecewise polynomial approximation and the spatial dimension (see Figure 6). Additional storage is needed for only one row of "ghost" elements (elements whose values must be obtained from a neighboring processor) along each edge of the processor's uniform subdomain. Thus, the size of the problems solved can be scaled easily with the number of processors.

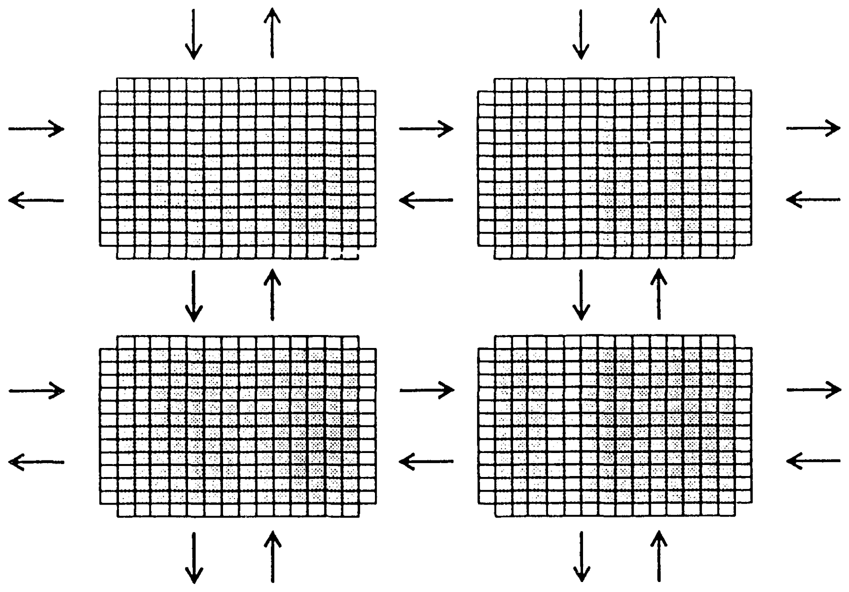


Figure 6. Communication patterns for the two-dimensional method on a parallel distributed memory computer. The small stencil requires only nearest-neighbor communication and only one row of "ghost" cells on each edge of the processors' subdomains.

We measure the performance of our method on an NCUBE/2 hypercube computer by considering the method's *scaled parallel efficiency*, the ratio of uniprocessor execution time for a problem of size W to execution time on N processors for a problem of size NW [10]. Thus, the amount of work per processor is kept constant as more processors are used, and we expect the solution time to be constant for each trial. Problem size is determined by the number of elements, number of processors, and degree of the approximating polynomials. In two dimensions,

$$W = \text{number of elements} \times \text{number of timesteps} \times (K+1)^2. \tag{19}$$

Example 5. In Table 2, we show the execution time for the two-dimensional spatially periodic problem

$$u_t + u_x + u_y = 0, \; t > 0, \tag{20a}$$

$$u(x, y, 0) = \sin(\pi x)\sin(\pi y), \tag{20b}$$

using various numbers of processors with $p = 2$ fixed. Each processor's subdomain contained 128 elements, and the problem was solved for 46 time steps. As indicated, the solution times increase only slightly with the dimension of the hypercube, demonstrating the high parallel efficiency of the method. We also show the ratio of the average execution time on all the processors to the maximum execution time among the processors. An average/maximum processor work ratio equal to one indicates perfect processor load balance. The average/maximum processor work ratio is above 0.98 for all hypercube dimensions due to the natural load balance of the non-adaptive method.

Number of Processors	Work (W)	Execution Time (secs.)	Parallel Efficiency	Avg/Max Processor Work Ratio
1	588,800	268.96		1.000
2	1,177,600	276.39	97.3%	.998
4	2,355,200	276.77	97.2%	.992
8	4,710,400	276.79	97.2%	.998
16	9,420,800	276.80	97.2%	.997
32	18,841,600	276.80	97.2%	.996
64	37,683,200	276.80	97.2%	.988
128	75,366,400	276.84	97.2%	.995
256	150,732,800	276.80	97.2%	.995
512	301,465,600	276.80	97.2%	.993
1024	602,931,200	276.80	97.2%	.995

Table 2. Scaled parallel efficiency for Example 5. Times were measured on the NCUBE/2.

6. A Tiling Approach to Dynamic Load Balancing

While the non-adaptive method exhibits near-perfect scaled parallel efficiency, processor load imbalances degrade the parallel performance of the adaptive p-refinement method. Non-uniform and changing processor work loads make dynamic load balancing necessary. Tiling is a modified version of the global load balancing technique developed by Leiss and Reddy [15]. The Leiss/Reddy load balancing technique uses local balancing performed within overlapping processor neighborhoods to achieve global load balance. A neighborhood is defined as a processor at the center of a circle of some predefined radius and all other processors within the circle. Each processor can be a neighborhood center. An example, showing 25 processors in 10 neighborhoods, is shown in Figure 7. Processors within a given neighborhood are balanced with respect to each other using local (as opposed to global) performance measurements. Individual processors may belong to several neighborhoods. In the Leiss/Reddy technique, work can be migrated from a processor to any other processor within the same neighborhood. Our technique restricts work migration so that elements are migrated only to neighboring processors following the element inter-connections.

In tiling, a processor's load depends on the number of elements that are assigned to its local mesh and the per-element processing cost. When each processor has the same number of elements and all element

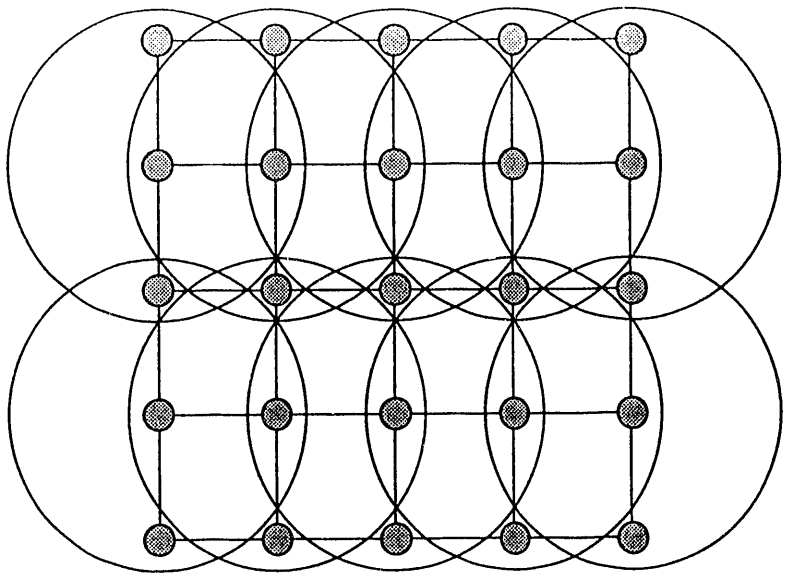


Figure 7. Neighborhood example: 25 processors in 10 neighborhoods

costs are equal, as in Example 5, global load balance results. When one processor is assigned elements having higher per-element processing costs than the elements assigned to another processor, as in the adaptive p-refinement method, processing imbalance is created.

Figure 8 illustrates the dynamic balancing provided by tiling. Without *a priori* knowledge, the data set is divided evenly among 16 tiles. After some period of processing where processors (0,1) and (3,2) were more utilized than their neighbors, processor (0,0) receives some of the data originally allocated to processor (0,1), and processor (3,2) gives processor (3,3) some of its data, as shown in Step 1. Processors (0,0) and (0,1) are now equally balanced yet out of balance with other processors. Thus, in Step 2, some data is migrated from processor (0,1) to processor (1,1). The ripple effect continues to move through processors (2,1) and (3,1) during subsequent balancing steps.

7. Tiling Algorithm Overview

To be integrated into the tiling environment, application programs are partitioned into two phases: a computation phase and a balancing phase. The *computation phase* corresponds to the application's implementation without load balancing. Each processor operates on its local data, exchanges inter-processor boundary data via ghost cells, and then processes the inter-processor boundary data. These actions are repeated until the application meets its termination condition.

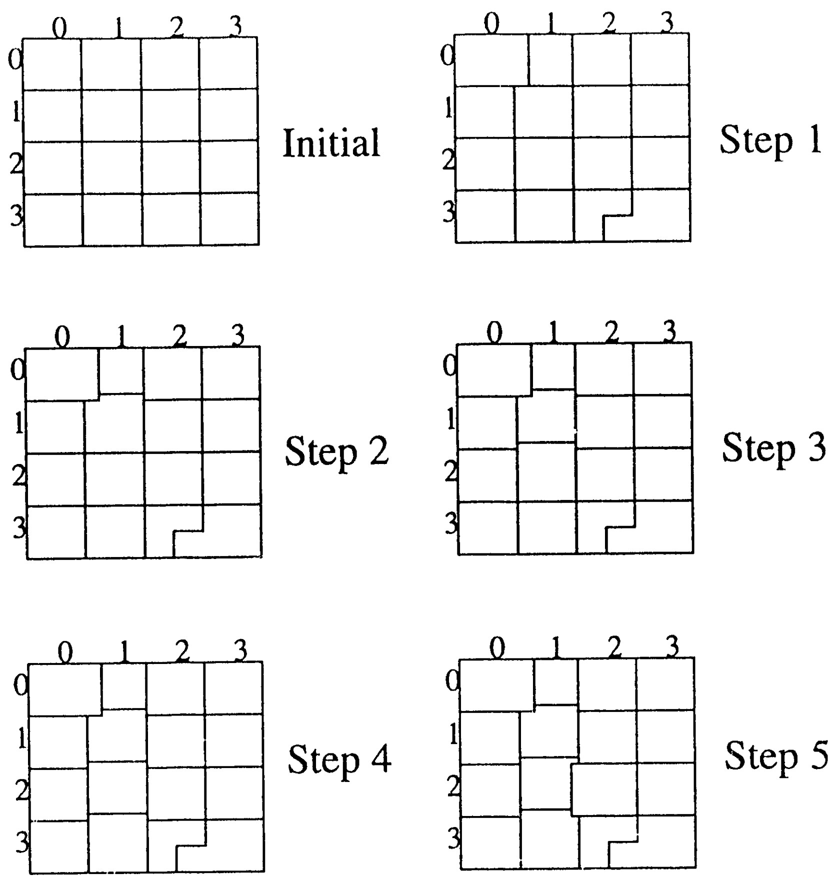


Figure 8. Migration Example

The tiling algorithm introduces dynamic load balancing in the inter-processor boundary data exchange operation. Some number of complete computation phases are performed followed by the balancing phase. Once the balancing phase has completed, another set of computation phases is performed. Each balancing phase consists of the following operations:

Determine work loads. Each processor determines its work load, the accumulated time taken to process its own local element data. This time is the time since the previous balancing phase less the time taken to exchange the inter-processor boundary data during the computation phase. Neighborhood average work loads are also calculated.

Determine from which processor to request work. Each processor compares its own work load to the work load of the other processors in its neighborhood. It looks for processors that have greater work loads than its own. If any are found, it selects the one with the greatest work load (ties are broken arbitrarily) and sends a request for work to that processor. Each processor may send only one work request, yet a single processor may receive several work requests.

Determine which processor(s) to grant work to. Each processor prioritizes the work requests it receives in order of size. Requests are satisfied until the processor's work load equals the neighborhood average.

Select export elements. Each exporting processor determines which elements to export to the processors that it has determined should receive work. Each element is assigned an exporting priority, initially zero. The priority is increased by 1 for each neighboring element in the element's processor, increased by 2 for each neighboring element in the importing processor, and decreased by 2 for each neighboring element in a foreign processor other than the importing processor. In this way, elements are "peeled" off the processor boundary in an attempt to prevent the creation of narrow, deep holes in the tile. This scheme determines an element's priority in a completely local manner, by examining pointers within the element's data structure to neighbors and ghost cells.

Notification and transfer of elements. Once the elements to be exported have been selected, the importing processors and those processors with ghost cells for the elements being migrated are notified. Importing processors then allocate space for the incoming elements, and the elements are transferred.

Each processor knows how many computation phases to perform before entering the balancing phase. Furthermore, the synchronous nature of the applications guarantees that each processor will enter the balancing phase at the same iteration as the other processors.

8. Tiling Application Interface

The tiling system is designed to be independent of the application. As shown in Figure 9, the Element Management System (EMS) uses three application interface routines: *App_preproc()*, *App_compute()*, and *App_postproc()*. The application programmer provides these routines, using the data structures for the element mesh provided by the EMS.

App_preproc() is called to create elements and assign them initial values. The EMS provides three major routines to the application pre-processing code: *create_ghost_element()*, *create_local_element()*, and *convert_links()*. The two element creation routines allocate element control structures and insert them into

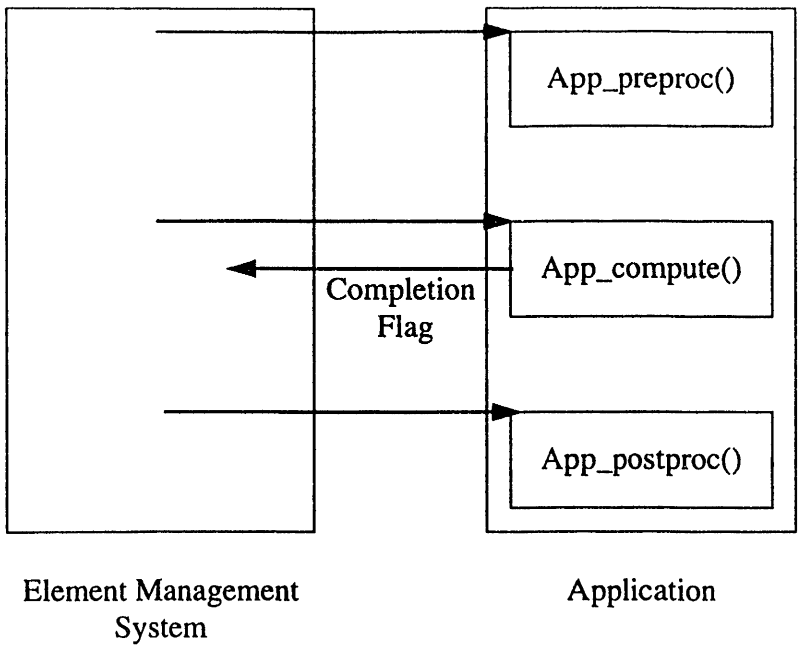


Figure 9. Application interface to the Element Management System

the appropriate element management trees. During element creation, element pointers are initialized to the neighboring elements' identification numbers, since the addresses of the elements may not be known yet. However, once all of the local and ghost elements have been created, the application code calls the *convert_links()* routine to replace the element numbers with actual pointers. *App_preproc()* also initializes application specific parameters and element data. The element trees are visible to the application programmer for this purpose.

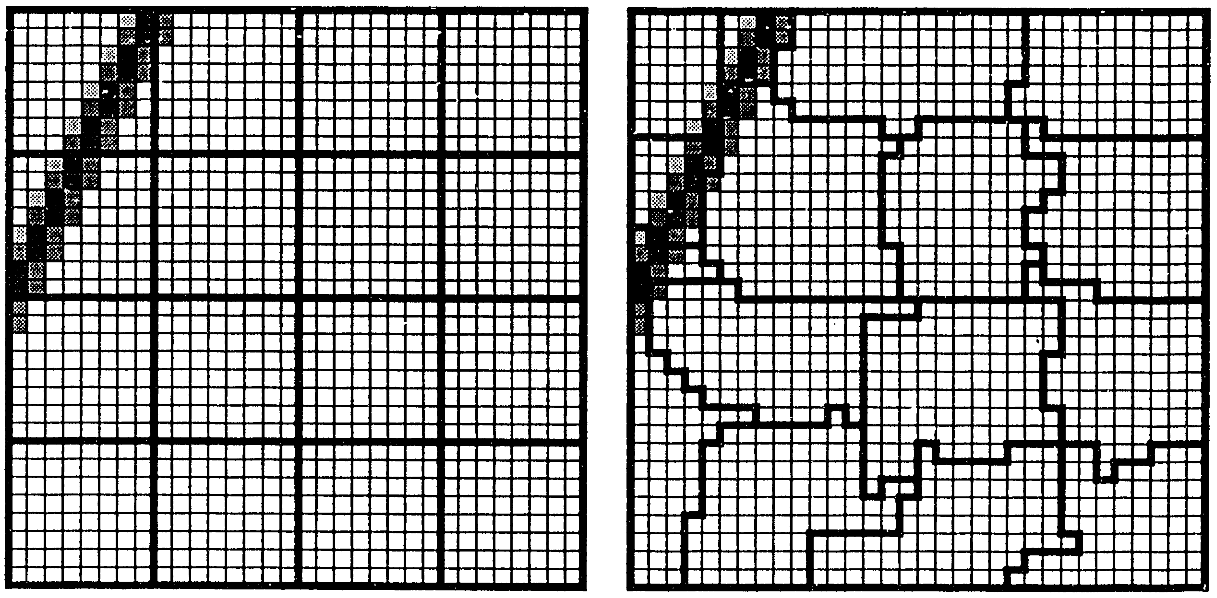
The processing performed by *App_preproc()* is expected to be quite unbalanced. For example, typically only one processor accesses host files to get run-time initialization information and broadcasts the information to the rest of the processor array. Therefore, migration processing is not performed during the application initialization stage.

App_compute() performs the application's computations. *App_compute()* is called iteratively until it returns a completion flag indicating application termination. The application programmer can schedule any processing desired in *App_compute()*; however, any communication done during *App_compute()* will affect the processor load calculation. The *App_compute()* processing time determines the processors' loads for the balancing phase.

App_postproc() is provided to separate the application's computation from the application's post-processing. Using this routine, the application may perform operations, such as transferring results to the host processor, without interference from the migration processing. Once *App_postproc()* returns, the application is terminated.

For our adaptive p-refinement finite element method, *App_preproc()* reads the input parameters file, broadcasts the input to the processor array, allocates memory for the elements, and calculates the initial values using (5b). *App_compute()* performs the Runge-Kutta integration of (5) and the adaptive p-refinement procedures. *App_postproc()* writes the solution at time t_{final} to the host processor.

Example 6. We solve (18) using the adaptive p-refinement method and tiling on 32×32-mesh on 16 processors of the nCUBE/2 hypercube. In Figure 10, we show the processor domain decomposition with and without balancing after 10 time steps. The shaded elements have higher-degree approximations and thus, higher work loads. The tiling algorithm redistributes the work so that processors with high-order elements have fewer elements than those processors with low-order elements. The total processing time was reduced 25.9% from 29.49 seconds to 21.86 seconds by balancing once each time step. The average/maximum processor work ratio without balancing is 0.353; with balancing, it is 0.609.



□ $p = 0$ ▨ $p = 2$ ▩ $p = 3$ ■ $p = 4$

Figure 10. Processor domain decomposition after 10 time steps for Example 6 using adaptive p-refinement on 16 processors without load balancing (left) and with load balancing (right). Shaded squares are high-order elements, and dark lines represent processor subdomain boundaries.

Example 7. We solve (18) for 229 time steps on 1024 processors of the nCUBE/2 without balancing and with balancing once each time step. With balancing, the maximum computation time (not including communication or balancing time) was reduced by 56.2% (see Table 3). The irregular subdomain boundaries created by the tiling algorithm increased the average communication time by 39%. Despite the extra communication time and the load balancing time, however, we see a 25.1% improvement in the total execution time.

	Without Balancing	With Balancing
Maximum Execution Time	596.56 secs.	446.60 secs.
Maximum Computation Time	544.37 secs.	238.27 secs.
Average Communication Time	80.99 secs.	112.65 secs.
Maximum Balancing Time	0.0 secs.	39.99 secs.
Average/Maximum Work Ratio	0.405	0.929

Table 3. Performance comparison for Example 7 without and with load balancing once each time step.

In Figures 11-14, we show the convergence of the processor work loads from uniform domain decomposition toward global balance. In Figure 11, we divide the difference of the maximum and minimum processor work loads at time $t = 0$ into 32 bins. As time progresses and the processors approach global balance, the processor distribution should converge toward one bin. The dashed line represents the number of processors in each bin at time $t = 0$. The solid line represents the number of processors in each bin after 20 balancing steps. At time $t = 0$, the maximum number of processors in a single bin is 904. After 20 time steps, the maximum number of processors in a single bin drops to 854 as work from more heavily loaded processors is migrated to the less heavily loaded processors. After 160 time steps, 847 processors have converged to a single bin with fewer processors in the heavily loaded bins, as shown by the dashed line of Figure 12. Twenty time steps later, the processors have converged even further toward global balance, with 853 processors in a single bin and almost no processors in the heavily loaded bins, as shown by the solid line.

In Figure 13, we show the +1 and -1 standard deviation curves of the maximum computation time for each time step. Initially, the deviation is large, indicating the processors are far from global balance. The deviations quickly become smaller, indicating the processors rapidly approach global balance. In Figure 14, we show the 5th, 35th, median, 65th, and 95th-percentile processor loads. The steep downward slope of the 95th-percentile curve indicates significant improvement in the load balance early in the program's execution.

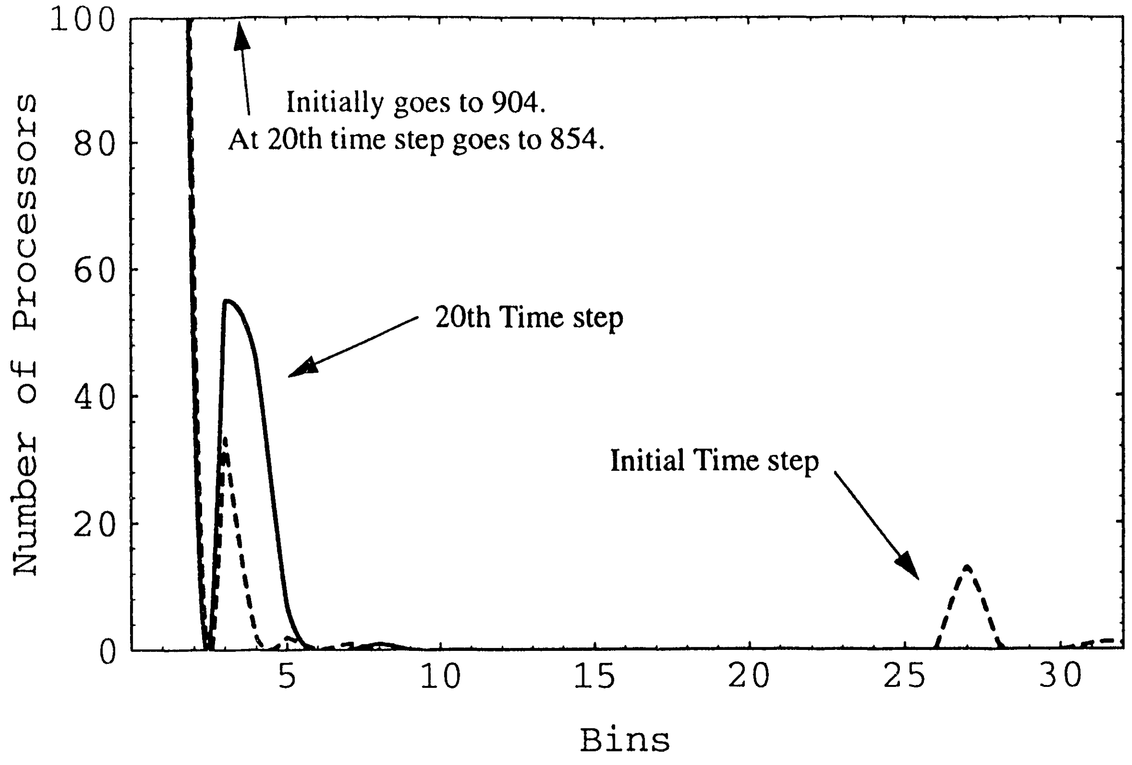


Figure 11. Processor work load distributions initially (dashed line) and after 20 time steps (solid line).

In Figure 15, we show the maximum processing costs per time step, including the computation time and the balancing time. The dashed line represents the maximum cost per time step without balancing; the solid line represents the maximum cost with balancing. Even including the load balancing time, the balanced computation's maximum cost per time step is significantly lower than without balancing. The spikes in both lines occur when the adaptive p-method's error tolerance was not satisfied in the time step, and the application "backed up" to recompute a higher-order approximation on the high error elements. Since relatively few elements are recomputed when the adaptive method backs up, the back-up time is extremely unbalanced. The tiling algorithm, however, does not use the back-up time to determine load balance. Balancing immediately after a back-up phase would move too much work away from the processors that needed to back up; some of the work would be migrated back to the processors in the next balancing phase. By disregarding the back-up time, the tiling algorithm avoids introducing such oscillations in the work load distribution.

In Figure 15, we show the cumulative maximum processing times with and without balancing. The immediate and sustained improvement of the application performance is shown.

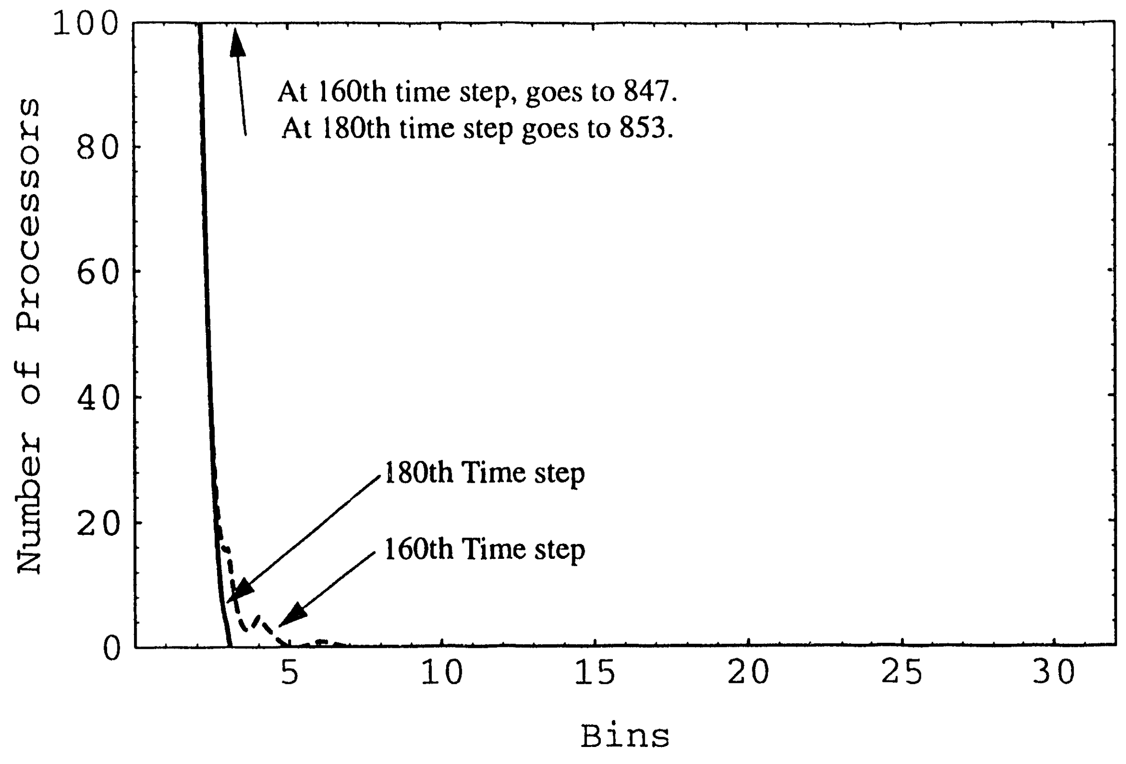


Figure 12. Processor work distributions after 160 time steps (dashed line) and after 180 time steps (solid line).

9. Conclusion

We have shown the viability of the discontinuous finite element method for solving systems of hyperbolic conservation laws in one and two dimensions on massively parallel computers. By not enforcing continuity at inter-element boundaries, we can accurately model problems with discontinuities. A projection limiter based on limiting the moments of the numerical solution eliminates oscillations near solution discontinuities while maintaining high-order accuracy near smooth extrema. Because of the compact stencil, the method (5, 18) can be parallelized on MIMD computers with scaled parallel efficiencies exceeding 90% and can model problems in complicated geometries more easily than traditional finite difference schemes. Adaptive p-refinement is used to solve problems to a user-specified accuracy with less computational expense than fixed-order computations. Using tiling to migrate data between processors, we can parallelize the adaptive p-refinement method with over 70% efficiency. Higher efficiency can be expected for longer runs, as the initial imbalance will be factored out.

In our future work, we will implement adaptive mesh (h-) refinement and combine the adaptive h- and p- techniques to obtain an adaptive hp-method that can optimize computational effort in both smooth

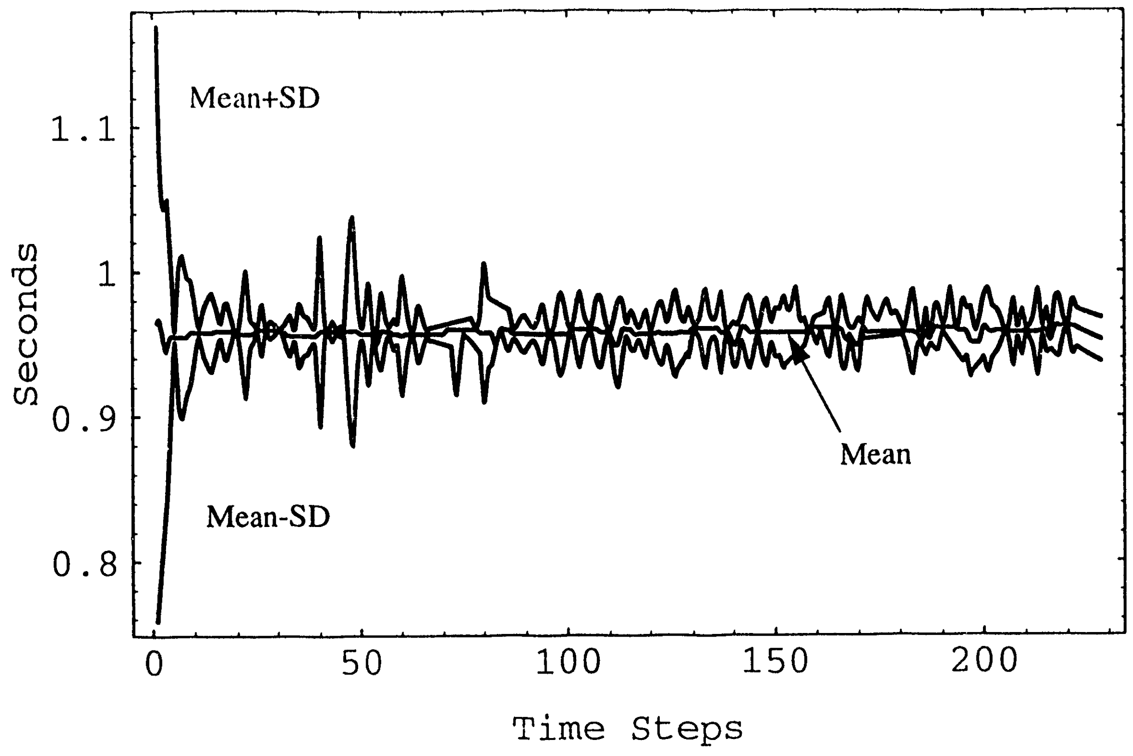


Figure 13. Processor work load mean and standard deviation for each time step.

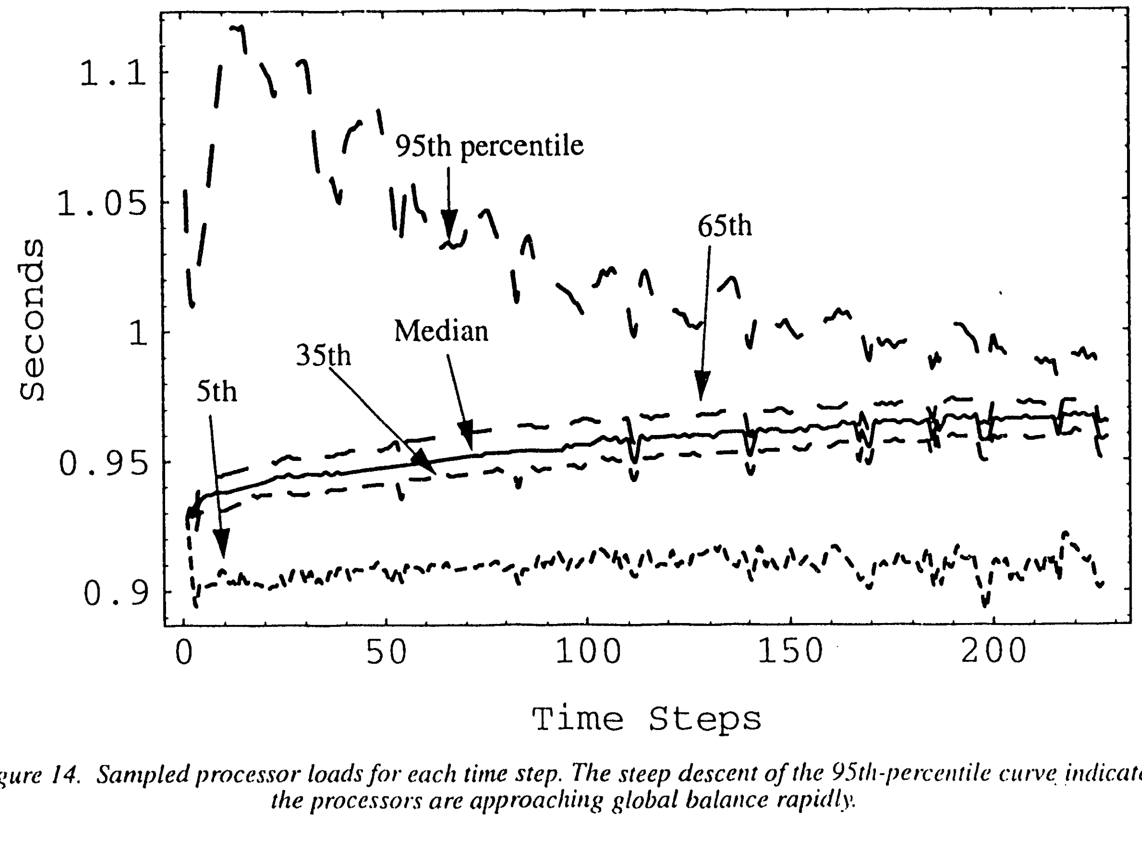


Figure 14. Sampled processor loads for each time step. The steep descent of the 95th-percentile curve indicates the processors are approaching global balance rapidly.

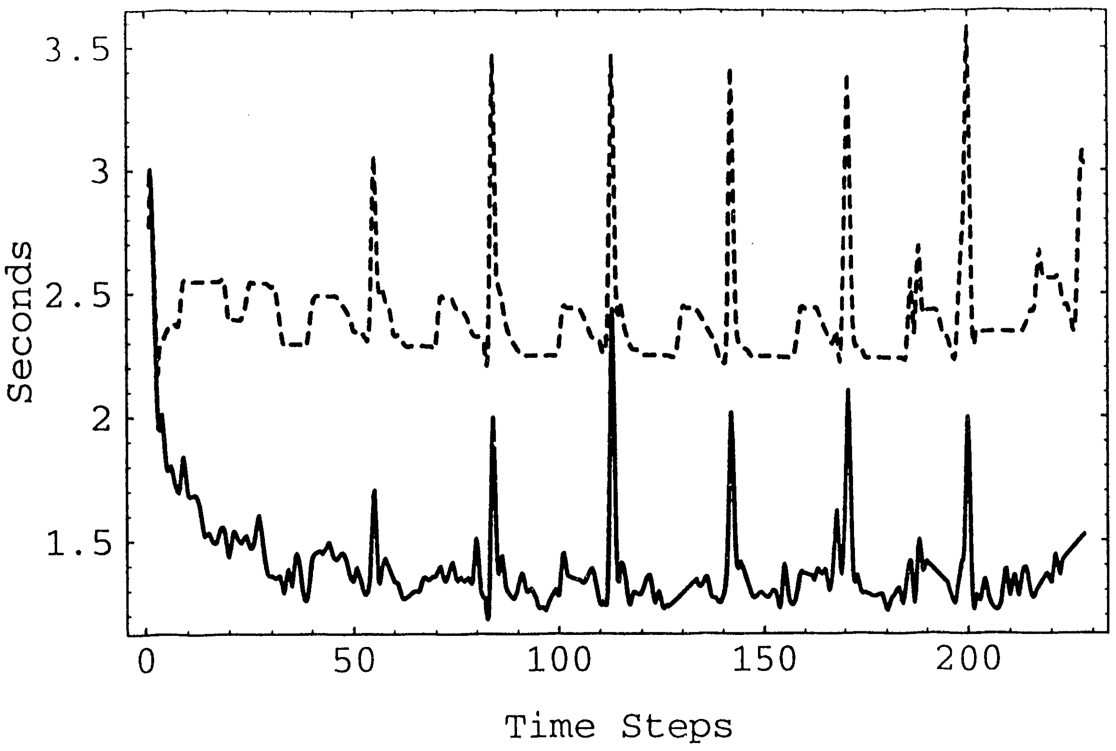


Figure 15. Maximum work load in each time step with balancing (solid line) and without balancing (dashed line).

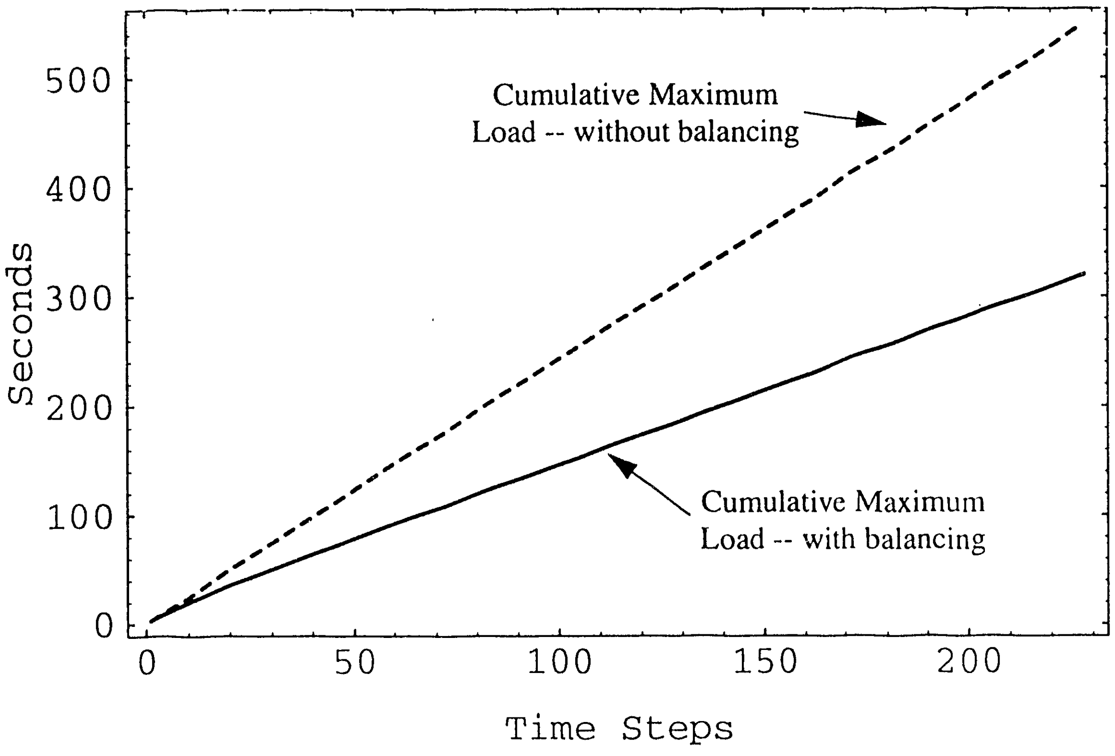


Figure 16. Cumulative maximum loads with and without balancing.

and discontinuous solution regions. The *hp*-method will present a serious challenge to the tiling algorithm, as the processor work loads and communication patterns become more complex.

As seen in Example 7, backing up when a time step is unacceptable in the adaptive method is very expensive and unbalanced. To reduce the amount of backing up necessary, we will use the pattern matching ideas of Bieterman, et al. [4] to predict which areas of the mesh should be refined in the next time step. We can also use the predicted mesh to estimate processor work loads in the next time step, and detect potential load imbalance. The tiling algorithm can then be used to migrate data before the time step computation begins, preventing the load imbalance.

10. References

[1] Adjerid, S., and J. E. Flaherty. "Second-Order Finite Element Approximations and a posteriori Error Estimation for Two-Dimensional Parabolic Systems." *Numer. Math.*, **53** (1988), 183-198.

[2] Adjerid, S., J. E. Flaherty, P. K. Moore, and Y. Wang. "High-Order Adaptive Methods for Parabolic Systems." *Physica-D* (1992) to appear.

[3] Arney, D.C. and J. E. Flaherty. "An Adaptive Local Mesh Refinement Method for Time-Dependent Partial Differential Equations." *App. Num. Math.*, 5 (1989), 257-274.

[4] Bieterman, M., J. Flaherty, and P. Moore. "Adaptive Refinement Methods for Non-Linear Parabolic Partial Differential Equations." *Accuracy Estimates and Adaptive Refinements in Finite Element Computations.* I. Babuska, et al., Eds. Wiley & Sons, (1986) 339-358.

[5] Biswas, R. "Parallel and Adaptive Methods for Hyperbolic Partial Differential Systems." Ph.D. dissertation. Rensselaer Polytechnic Institute, August, 1991.

[6] Biswas, R., K. Devine, and J. Flaherty, "Parallel, Adaptive Finite Element Methods for Conservation Laws." *Applied Numerical Mathematics*, to appear.

[7] Cockburn, B., S. Hou, and C.-W. Shu. "The Runge-Kutta Local Projection Discontinuous Galerkin Finite Element Method for Conservation Laws IV: The Multidimensional Case." *Math. Comp.*, **54** (1990), 545-581.

[8] Cockburn, B., S.-Y. Lin, and C.-W. Shu. "TVB Runge-Kutta Local Projection Discontinuous Galerkin Finite Element Method for Conservation Laws III: One-Dimensional Systems." *Jrnl. of Comp. Phys.*, **84** (1989), 90-113.

[9] Cockburn, B., and C.-W. Shu. "TVB Runge-Kutta Local Projection Discontinuous Galerkin Finite Element Method for Conservation Laws II: General Framework." *Math. Comp.*, 52 (1989), 411-435.

[10] Gustafson, J., G. Montry, and R. Benner. "Development of Parallel Methods for a 1024-Processor Hypercube." *SIAM Jrnl. Sci. Stat. Comp.* 9 (1988), 609-638.

[11] Hammond, S. *Mapping Unstructured Grid Computations to Massively Parallel Computers.* Ph.D. thesis, Rensselaer Polytechnic Institute, Dept. of Computer Science, Troy, NY, 1992.

[12] Hendrickson, B., and R. Leland. "Multidimensional Spectral Load Balancing." Sandia National Laboratories Tech. Rep. SAND93-0074.

[13] Kernighan, B. and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs." Bell Systems Tech. Jrnl., 29 (1970), 291-307.

[14] Lafon, F. and S. Osher. "High-Order Filtering Methods for Approximating Hyperbolic Systems of Conservation Laws." *ICASE* Report No. 90-25, March 1990.

[15] Leiss, E., and H. Reddy. "Distributed Load Balancing: Design and Performance Analysis." *W.M. Keck Research Computation Laboratory*. 5 (1989) 205-270.

[16] Rank, E. and I. Babuska. "An Expert System for the Optimal Mesh Design in the hp-Version of the Finite Element Method." *Intl. Jrnl. Num. Meth. in Engng.*, **24** (1987), 2087-2106.

[17] Richtmyer, R.D., and K.W. Morton. *Difference Methods for Initial Value Problems*, Interscience, New York, 1967.

[18] Shu, C.-W. "TVB Boundary Treatment for Numerical Solutions of Conservative Laws." *Math. Comp.*, **49** (1987), 123-134.

[19] Shu, C.-W., and S. Osher. "Efficient Implementation of Essentially Non-oscillatory Shock-Capturing Schemes, II." *Jrnl. of Comp. Phys.*, **83** (1989), 32-78.

[20] Sod, G. "A Survey of Several Finite Difference Methods for Systems of Nonlinear Hyperbolic Conservation Laws." *Jrnl. of Comp. Phys.*, **27** (1978), 1-31.

[21] Sweby, P.K. "High Resolution Schemes Using Flux Limiters for Hyperbolic Conservation Laws." *SIAM J. Numer. Anal.*, **21** (1984), 995-1011.

[22] Szabo, B. and I. Babuska. *Introduction to Finite Element Analysis*, Wiley, New York, 1990.

[23] Van Leer, B. "Towards the Ultimate Conservative Difference Scheme. IV. A New Approach to Numerical Convection." *Jrnl. of Comp. Phys.*, **23** (1977), 276-299.

[24] Wheat, S. *A Fine Grained Data Migration Approach to Application Load Balancing on MP MIMD Machines.* Ph.D. Thesis. Univ. of New Mexico, Dept. of Computer Science, Albuquerque, NM, 1992.

DATE
FILMED
10/13/93

END