A Parallel Adaptive *hp*-Refinement Finite Element Methods with Dynamic Load Balancing for the Solution of Hyperbolic Conservation Laws*

Karen D. Devine† and Joseph E. Flaherty§

Abstract

We describe an adaptive *hp*-refinement local finite element procedure for the parallel solution of hyperbolic systems of conservation laws on rectangular domains. The local finite element procedure utilizes spaces of piecewise-continuous polynomials of arbitrary degree and coordinated explicit Runge-Kutta temporal integration. A solution limiting procedure produces monotonic solutions near discontinuities while maintaining high-order accuracy near smooth extrema. A modified tiling procedure maintains processor load balance on parallel, distributed-memory MIMD computers by migrating finite elements between processors in overlapping neighborhoods to produce locally balanced computations. Grids are stored in tree data structures, with finer grids being offspring of coarser ones. Within each grid, AVL trees simplify the transfer of information between neighboring processors and the insertion and deletion of elements as they migrate between processors. Computations involving Burgers' and Euler's equations of inviscid flow demonstrate the effectiveness of the *hp*-refinement and balancing procedures relative to non-balanced adaptive and balanced non-adaptive procedures.

1. Introduction

In recent years, adaptive methods for solving partial differential equations have been growing in popularity [8]. With problems becoming more complex through multi-dimensional and nonlinear effects, the automation provided by adaptivity provides, perhaps, the only reasonable way to address them. Early adaptive techniques of mesh motion (*r*-refinement) have been giving way to methods that combine mesh refinement/coarsening (*h*-refinement) with order variation (*p*-refinement) [8]. In this regard, adaptivity for partial differential equations is following the path of similar strategies used for ordinary differential equations [14].

As advances in computer architecture enable the solution of complex three-dimensional problems, the efficiency, reliability, and robustness provided by adaptivity will make its use even more advantageous. Parallel computation will be essential in these simulations. Thus, our aim is to show that sophisticated adaptive algorithms involving *hp*-refinement can be developed and that they perform well in parallel environments. Although the complex logic and localized computation used within adaptive strategies would seem to be at odds with efficient parallel computation, several recent results demonstrate that this is not the case. Berger [3] has developed local *h*-refinement finite difference procedures for conservation laws on

* This work was supported by the United States Department of Energy under Contract DE-AC04-94AL85000, Sandia National Laboratories under Research Agreement AD-9585, and the U.S. Army Research Office under Contract DAAH04-95-1-0091.

† Parallel Computing Sciences Department, Sandia National Laboratories, Albuquerque, NM 87185-1109.

§ Department of Computer Science and Scientific Computation Research Center, Rensselaer Polytechnic Institute, Troy, NY 12180.

data-parallel computers. Biswas et al. [6] and de Cougny et al. [11] describe adaptive h- and p-refinement local finite element procedures for conservation laws on distributed-memory MIMD computers. Work is advancing to three dimensions with Bottaso et al. [7] and Kallinderis [15] describing h-refinement procedures for unstructured-mesh computation for Euler flows on distributed-memory MIMD computers.

Little has been done with parallel hp-refinement aside from the dissertations of Bey [5] and Devine [12], who both consider local finite element procedures [9, 10] for conservation laws on distributed-memory MIMD computers. Following Devine [12], we describe a local finite element hp-refinement procedure for the parallel solution of conservation laws on two-dimensional rectangular domains. An adaptive limiting procedure (Section 2) preserves monotonicity near discontinuities and serves as an indicator to determine whether h- or p-refinement is appropriate. Generally, p-refinement is more efficient in regions where solutions are smooth while h-refinement is the choice near singularities [1]. The need for solution limiting indicates the presence of solution discontinuities and, hence, the preference of h-refinement relative to p-refinement. By using less limiting than other approaches [10], the adaptive limiting also preserves high order near smooth extrema [6].

Our method uses explicit time integration with order and step-size keyed to the corresponding spatial quantities. As is typical with local refinement methods [4], data structures include a tree of (uniform) grids, with finer grids stored as offspring of the coarse grids in which they lie. Thus, time steps on grids closer to the tree root are larger than those at deeper tree levels. The adaptive strategy uses estimates of the local discretization error computed by p-refinement to indicate the need for enrichment or coarsening (Section 3). Order variation by unit amounts is done prior to h-refinement.

Processor load-balancing must be dynamic since frequent adaptive enrichment will upset a balanced computation. Extending a "tiling" procedure for uniform-mesh finite difference computation [24] that was based on a global optimization procedure of Leiss and Reddy [17, 18], balance is restored by performing a local optimization on overlapping processor "neighborhoods" (Section 5). Each processor's neighborhood consists of those processors having elements sharing edges with those in the defining processor. Local balance is achieved by migrating elements between processors in the neighborhood.

Balance is complicated by the heterogeneous method orders and by meshes at different tree levels. Load-balancing is applied separately at each mesh level (Section 4). Thus, elements introduced by refinement may require migration to processors other than those of their parent, increasing the amount of

communication needed between mesh levels. However, this increase in communication cost is typically offset by an improved load balance. Tree data structures are created to simplify the insertion and deletion of elements in grids during migration and to reduce communication overhead between neighboring processors (Section 6). With transient problems, one iteration of the tiling procedure is performed per time step; thus, a perfect balance is generally not achieved when adaptive refinement is used. However, since adaptive refinement is highly localized, a fast, incremental load-redistribution strategy is preferable to a more expensive, global strategy.

Examples involving Burgers' equation and the Euler equations of inviscid flow indicate that the adaptive *hp*-refinement solutions with balancing are almost five times more efficient than fixed-mesh, fixed-order solutions having comparable accuracy (Section 7). The experiments demonstrate the performance of the adaptive refinement and load-balancing strategies on both the nCUBE/2 (hypercube-architecture) and Intel Paragon (mesh-architecture) parallel computers.

2. Local Finite Element Formulation

We consider systems of two-dimensional hyperbolic conservation laws in m variables having the form

$$\mathbf{u}_t + \mathbf{f}(\mathbf{u})_x + \mathbf{g}(\mathbf{u})_y = 0, \ (x, y) \in \Omega, \ t > 0, \tag{1a}$$

with the initial conditions

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

and appropriate well-posed boundary conditions on $\partial\Omega$. The subscripts t, x, and y denote partial differentiation with respect to time and the spatial coordinates, and $\mathbf{u}$, $\mathbf{u}^0$, $\mathbf{f}$, and $\mathbf{g}$ are m-vectors on the problem domain $\Omega \times (t > 0)$.

Restrict Ω to be rectangular and partition it into rectangular elements

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

Construct a weak form of (1) on Ω_{ij} by multiplying (1a) by a test function $\mathbf{v} \in L^2(\Omega_{ij})$, integrating the result over Ω_{ij}, and using the divergence theorem [6]. Mapping the integrals on Ω_{ij} to $\Omega_C = \{ (\xi, \eta) \,|\, -1 \le \xi, \eta \le 1 \}$, approximating $\mathbf{u}(x, y, t)$ on Ω_C by a bi-p^{th}-degree polynomial

$$\mathbf{u}(\xi, \eta, t) \approx \mathbf{U}_{ij}(\xi, \eta, t) = \sum_{k=0}^{p} \sum_{l=0}^{p} \mathbf{c}_{ijkl} P_k(\xi) P_l(\eta), \ (\xi, \eta) \in \Omega_C, \tag{3}$$

testing against functions proportional to the product of Legendre polynomials $P_r(\xi)P_s(\eta)$, and using the properties of Legendre polynomials [6, 12, 22] yields

$$\dot{\mathbf{c}}_{ijkl} = -\frac{(2k+1)\ (2l+1)}{2\Delta x_i \Delta y_j} (\mathbf{I}_1 + \mathbf{I}_2 + \mathbf{I}_3), \ k, l = 0, 1, \ldots, p, \ t > 0, \tag{4a}$$

where

$$\mathbf{I}_1 = -\int_{-1}^{1}\int_{-1}^{1} [\Delta y_j P_k'(\xi) P_l(\eta)\mathbf{f}(\mathbf{U}_{ij}) + \Delta x_i P_k(\xi) P_l'(\eta)\mathbf{g}(\mathbf{U}_{ij})]\, d\xi d\eta, \tag{4b}$$

$$\mathbf{I}_2 = \Delta y_j \int_{-1}^{1} [P_l(\eta)\mathbf{f}(\mathbf{U}_{ij}(1, \eta, t)) - (-1)^k P_l(\eta)\mathbf{f}(\mathbf{U}_{ij}(-1, \eta, t))]\, d\eta, \tag{4c}$$

$$\mathbf{I}_3 = \Delta x_i \int_{-1}^{1} [P_k(\xi)\mathbf{g}(\mathbf{U}_{ij}(\xi, 1, t)) - (-1)^l P_k(\xi)\mathbf{g}(\mathbf{U}_{ij}(\xi, -1, t))]\, d\xi, \tag{4d}$$

$$\Delta x_i = x_i - x_{i-1}, \tag{4e}$$

$$\Delta y_j = y_j - y_{j-1}. \tag{4f}$$

The initial Galerkin coordinates are determined by L^2-projection as

$$\mathbf{c}_{ijkl}(0) = \frac{(2k+1)\ (2l+1)}{4} \int_{-1}^{1}\int_{-1}^{1} P_k(\xi) P_l(\eta)\mathbf{u}^0(x(\xi), y(\eta)) d\xi d\eta, \ k, l = 0, 1, \ldots, p. \tag{4g}$$

Integrals are evaluated using $(p+1)^2$-point Gaussian quadrature, and the ordinary differential system (4) is solved on Ω by explicit Runge-Kutta integration of order p.

The boundary fluxes $\mathbf{f}(\mathbf{U}_{ij}(\pm 1, \eta, t))$ and $\mathbf{g}(\mathbf{U}_{ij}(\xi, \pm 1, t))$ remain unspecified on $\partial\Omega_{ij}$ with (4) since the approximate solution is discontinuous there. We specify them using a "numerical flux" function $\mathbf{h}(\mathbf{U}_{ij}^+, \mathbf{U}_{ij}^-)$ dependent on solution states $\mathbf{U}_{ij}^+$ and $\mathbf{U}_{ij}^-$ on the inside and outside, respectively, of $\partial\Omega_{ij}$. We have used the local Lax-Friedrichs numerical flux

$$\mathbf{h}(\mathbf{U}_{ij}^{+}, \mathbf{U}_{ij}^{-}) = \frac{1}{2}[\mathbf{f}(\mathbf{U}_{ij}^{+}) + \mathbf{f}(\mathbf{U}_{ij}^{-}) - \alpha(\mathbf{U}_{ij}^{-} - \mathbf{U}_{ij}^{+})], \tag{5a}$$

$$\alpha = \max_{1 \le l \le m}(|\lambda_l(\mathbf{f}_{\mathbf{u}}(\mathbf{U}_{ij}))|), \ \mathbf{U}_{ij}^{+} \le \mathbf{U}_{ij} \le \mathbf{U}_{ij}^{-}, \tag{5b}$$

where $\lambda(\mathbf{f}_{\mathbf{u}})$ is an eigenvalue of the Jacobian $\mathbf{f}_{\mathbf{u}}$. The boundary fluxes $\mathbf{g}(\mathbf{U}_{ij}(\xi, \pm 1, t))$ are approximated similarly.

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

$$h = \max_{\substack{i = 1, 2, \ldots, I \\ j = 1, 2, \ldots, J}} (\Delta x_i, \Delta y_j), \tag{6}$$

convergence expected in L^1 for a p^{th}-degree approximation [10]. When $p > 0$, flux or solution limiting is needed to prevent spurious oscillations near solution discontinuities. With flux limiting, $\mathbf{h}(\mathbf{U}_{ij}^{+}, \mathbf{U}_{ij}^{-})$ is restricted in order to obtain a monotone solution [20, 21, 23]. In the local finite element method, flux limiters preserve monotonicity of solution averages, $\mathbf{c}_{ij00}$, $i = 1, 2, \ldots I$, $j = 1, 2, \ldots, J$, but do not modify the higher-order coefficients to preserve monotonicity of the finite element solution.

We have had greater success with solution limiting, where $\mathbf{U}_{ij}$, $i = 1, 2, \ldots I$, $j = 1, 2, \ldots, J$, is restricted after each Runge-Kutta stage to eliminate oscillations. Cockburn and Shu [10] describe a procedure for the limiting of scalar problems that prevents the approximate solution at any point within the element from taking values outside of the range spanned by the neighboring solution averages. While preserving monotonicity of the average numerical solution, the limiting flattens solutions near smooth extrema so that high-order accuracy is lost there. To overcome this, we extend the one-dimensional limiting procedure of Biswas et al. [6] to two dimensions by limiting in each coordinate direction. This procedure [6] essentially limits derivatives of the solution within an element with respect to their approximations over neighboring elements. Assuming a scalar problem and differentiating (3), we find

$$\begin{aligned} \frac{\partial^r}{\partial \xi^r} U_{ij}(\xi, \eta, t) = & \left(\prod_{k=0}^{r} |2k-1| \right) \sum_{l=0}^{p} c_{ijrl}(t) P_l(\eta) \\ & + \sum_{k=r+1}^{p} \sum_{l=0}^{p} c_{ijkl}(t) P_l(\eta) \frac{\partial^r}{\partial \xi^r} P_k(\xi), \quad r = 1, 2, \ldots, p, \quad (\xi, \eta) \in \Omega_C. \end{aligned} \tag{7}$$

To limit the r^{th}-degree coefficients c_{ijrl}, $r = 1, 2, \ldots, p$, $l = 0, 1, \ldots, r$, we examine the r^{th} derivative within Ω_{ij} of the r^{th}-degree polynomial

$$\hat{U}_{ij}(\xi, \eta, t) = \sum_{k=0}^{r} \sum_{l=0}^{r} c_{ijkl}(t) P_k(\xi) P_l(\eta), \; r = 1, 2, \ldots, p, \; (\xi, \eta) \in \Omega_C. \tag{8}$$

Applying (7) to (8), we have

$$S_{ij}(\eta, t) \equiv \frac{\partial^r}{\partial \xi^r} \hat{U}_{ij}(\xi, \eta, t) = \left(\prod_{k=0}^{r} |2k-1| \right) \sum_{l=0}^{r} c_{ijrl}(t) P_l(\eta). \tag{9}$$

We evaluate (9) at $r+1$ uniformly spaced points $\eta_s = -1 + 2s/r$, $s = 0, 1, \ldots, r$, on the element edges, $\xi = \pm 1$, and compare the result with approximations over neighboring elements obtained by taking the differences of the average values of the $(r-1)^{st}$-derivative with respect to ξ in $\Omega_{i+1,j}$, Ω_{ij}, and $\Omega_{i-1,j}$. Using (7) and the properties of Legendre polynomials [22], the average derivative in Ω_{ij} is

$$A_{\xi ij}(t) \equiv \frac{1}{4} \int_{-1}^{1} \int_{-1}^{1} \frac{\partial^{r-1}}{\partial \xi^{r-1}} U_{ij}(\xi, \eta, t) d\xi d\eta = \left(\prod_{k=0}^{r-1} |2k-1| \right) c_{ij,\, r-1,\, 0}(t). \tag{10}$$

We limit (9) with respect to differences of (10) as

$$S_{ij}(\eta_s, t) = minmod(S_{ij}(\eta_s, t), \tfrac{1}{2}(A_{\xi,\, i+1,\, j} - A_{\xi ij}), \tfrac{1}{2}(A_{\xi ij} - A_{\xi,\, i-1,\, j})), \; s = 0, 1, \ldots, r, \tag{11a}$$

where

$$minmod(a, b, c) = \begin{cases} \operatorname{sgn}(a) \min(|a|, |b|, |c|), & \text{if } \operatorname{sgn}(a) = \operatorname{sgn}(b) = \operatorname{sgn}(c), \\ 0, & \text{otherwise,} \end{cases} \tag{11b}$$

and redetermine c_{ijrl}, $l = 0, 1, \ldots, r$, using (9) with $\eta = \eta_s$, $s = 0, 1, \ldots, r$.

This procedure is applied analogously in the η-direction to limit c_{ijkr}, $r = 1, 2, \ldots, p$, $k = 0, 1, \ldots, r$. The coefficient c_{ijrr} is uniquely specified by limiting (11) in both the ξ- and η-directions and using

$$c_{ijrr}(t) = minmod(c^{\xi}_{ijrr}(t), c^{\eta}_{ijrr}(t)), \tag{11c}$$

where c^{ξ}_{ijrr} and c^{η}_{ijrr} are the values of c_{ijrr} determined by the limiting (11) in the ξ- and η- directions, respectively.

The two-dimensional limiting procedure is applied adaptively. First, c_{ijpk} and c_{ijkp}, $k = 0, 1, \ldots, p$, are limited. If they are changed by the limiter, $c_{ij,p-1,k}$ and $c_{ijk,p-1}$, $k = 0, 1, \ldots, p-1$, are limited. Lower-order coefficients are successively limited when the next higher terms are changed by the limiter. The high-order coefficients are limited again using the updated lower-order terms. In this way, the limiting is applied only where it is needed, and high-order accuracy is retained in smooth regions.

For vector systems, the scalar limiting function can be applied component-wise; however, Cockburn et al. [9] 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 (1a) [9, 16]. The diagonalizing matrices $\mathbf{T}(\mathbf{u})$ and $\mathbf{T}^{-1}(\mathbf{u})$ (consisting of the right and left eigenvectors of the Jacobian $\mathbf{f_u}$) are evaluated using the average values of $\mathbf{U}_{ij}$, $i = 1, 2, \ldots, I$, $j = 1, 2, \ldots, J$. The scalar projection limiter is applied in the ξ-direction to each field of the characteristic vector. The result is then projected back to the physical space by post-multiplication by $\mathbf{T}^{-1}(\mathbf{U}_{ij})$. Similar projections using the eigenvectors of $\mathbf{g_u}$ are applied for limiting in the η-direction.

3. Adaptive *hp*-Refinement

Adaptive *hp*-refinement yields improved computational efficiency by using high-order approximations in regions of the domain where solutions are smooth and refined meshes near solution discontinuities. A spatial error estimate is used to control order variation procedures that attempt to keep the global L^1-error less than a specified tolerance ε by maintaining

$$E_{ij}(t) \le TOL = \frac{\varepsilon}{IJ},\ i = 1, 2, \ldots, I,\ j = 1, 2, \ldots, J, \tag{12}$$

where E_{ij} is the maximum local L^1-error estimate of the solution vector on element Ω_{ij}. For these experiments, we use a p-refinement-based spatial error estimate where the local error is taken to be the difference between two approximations of differing degrees; thus,

$$E_{ij}(t) = \left\| \int_{\Omega_{ij}} \left| \mathbf{U}^{p+1}_{ij}(\mathbf{x}, t) - \mathbf{U}^{p}_{ij}(\mathbf{x}, t) \right| d\mathbf{x} \right\|_{\infty},\ i = 1, 2, \ldots, I,\ j = 1, 2, \ldots, J, \tag{13}$$

where the superscript identifies the degree of the approximation (3). For a two-dimensional problems, the estimate requires the solution of an additional $(p+2)^2$ ordinary differential equations (ODEs) per element. While this estimate is computationally expensive, it is less expensive than Richardson's extrapolation-(h-refinement-) based estimates [19], which require the solution of $4(p+1)^2$ additional ODEs per element. In addition, it can be used to reduce the effort involved in recomputing $\mathbf{U}_{ij}$ and its error estimate when p-refinement is needed. Other less expensive error estimation procedures are possible [5, 12, 13].

The adaptive hp-refinement strategy relies on the projection limiter to determine whether to perform h- or p-refinement in high-error regions. In regions where the high-order terms of the approximation are changed by the limiter, raising the degree of the approximation yields no benefit, and the mesh should be refined if greater accuracy is needed. In smooth-solution regions where little limiting is done, the degree of the approximation should be increased to benefit from the higher convergence rate of high-order methods. Following [1], in low-error regions, we coarsen the mesh if the solution is smooth and decrease the polynomial degree if it is not smooth.

Elements created by adaptive h-refinement are stored in a hierarchical tree of meshes [4]. The coarse base mesh is the root of the tree, and refined meshes are children in the tree. The local finite element method and adaptive hp-refinement strategy are applied recursively to each level of the tree of meshes. Refinement is performed in both space and time to maintain the ratio of the time-step size to the mesh spacing.

The adaptive hp-refinement algorithm is described in Figure 1. The solution and error estimate for the time step from t to $t+\Delta t$ are calculated on a base mesh. When enrichment is necessary, we first enrich the degree of the approximation on high-error elements in smooth regions by replacing $\mathbf{U}_{ij}^{p}(x, y, t+\Delta t)$ by $\mathbf{U}_{ij}^{p+1}(x, y, t+\Delta t)$, the $(p+1)$-degree approximation computed from the error estimate (13). We initialize the new error estimate $\mathbf{U}_{ij}^{p+2}(x, y, t)$ as $\mathbf{U}_{ij}^{p+1}(x, y, t)$ with the coefficients of the $(p+2)$-degree terms set to zero, and recompute only $\mathbf{U}_{ij}^{p+2}$ over the time step. Enrichment is repeated until no further p-refinement is needed.

High-error elements Ω_l in non-smooth regions are then divided into μ^2 fine elements, Ω_n, $n = 1, 2, \ldots, \mu^2$. Elements sharing edges or vertices with high-error elements are also refined as a buffer between high- and low-error regions and to maintain a difference of at most one mesh level across edges of refined regions. Solution values on the refined elements at time t are obtained by L^2-projection of the solution on Ω_l to Ω_n; thus,

```
void adaptive_hp_refinement(mesh, t_start, t_final, Δt, TOL)
{
    t = t_start;
    while (t < t_final) {
        perform_runge_kutta_time_step(mesh);
        do {
            Solution_Accepted = TRUE;
            for each element of mesh {
                error_estimate = calculate_estimate(element);
                if (error_estimate > TOL) {
                    if (smooth_region(element)) {
                        P_mark_element_for_p_enrichment(element);
                        Solution_Accepted = FALSE;
                    } else {
                        H_mark_element_for_h_refinement(element);
                        add_new_elements(fine_mesh);
                    }
                } else if (error_estimate < H_min * TOL) {
                    if (element_already_refined && smooth_region(element))
                        mark_element_for_coarsening(element);
                }
            } /* end for statement */
            if (!Solution_Accepted) {
                for each element of mesh
                    correct_one_degree_differences(element);
                increase_polynomial_degree_on_P_marked_elements();
                recalculate_solution_on_p_enriched_elements();
            }
        } while (!Solution_Accepted);
        if (mesh is refined) {
            buffer_H_marked_elements(fine_mesh);
            project_coarse_data_for_newly_refined_elements(mesh, fine_mesh);
            coarsen_marked_elements_and_remove_underlying_fine_elements();
            adaptive_hp_refinement(fine_mesh, t, t + Δt, Δt / μ, TOL / μ);
            interpolate_fine_solution_to_coarse_mesh(fine_mesh, mesh);
        }
        accept_solution(mesh);
        predict_degrees_for_next_time_step(mesh);
        t = t + Δt;
    } /* end while loop */
}
```

Figure 1. Algorithm for adaptive hp-refinement.

$$\mathbf{c}_{nrs}(t) = \frac{\int\int_{\Omega_n} P_r(\xi(x))P_s(\eta(y))\mathbf{U}_l(x, y, t)dydx}{\int\int_{\Omega_n} P_r^2(\xi(x))P_s^2(\eta(y))dydx}, \quad r, s = 0, 1, \ldots, p, \; n = 1, 2, \ldots, \mu^2. \tag{14}$$

The time step is also refined on the finer mesh, so that μ time steps of size $\Delta t_{fine} = \Delta t/\mu$ are taken on the fine mesh to arrive at time $t + \Delta t$. High-order temporal interpolation on coarse elements is used to compute numerical fluxes across coarse/fine mesh interfaces at intermediate Runge-Kutta stages on the fine mesh [12]. The adaptive hp-refinement algorithm is then recursively applied to the refined mesh level with time step Δt_{fine}. After μ time steps on a fine mesh level, solutions from fine elements Ω_n, $n = 1, 2, \ldots, \mu^2$, are interpolated to their coarse parent element Ω_l using L^2-projection; thus,

$$\mathbf{c}_{lrs}(t + \mu\Delta t_{fine}) = \frac{\sum_{n=1}^{\mu^2} \int\int_{\Omega_n} P_r(\xi(x))P_s(\eta(y))\mathbf{U}_n(x, y, t + \mu\Delta t_{fine})dydx}{\int\int_{\Omega_l} P_r^2(\xi(x))P_s^2(\eta(y))dydx}, \quad r, s = 0, 1, \ldots, p. \tag{15}$$

Simplification of the integrals in (14) and (15) is achieved using the properties of Legendre polynomials [12, 22].

To reduce the overhead of creating new refined elements at each base mesh time step, we retain the refined meshes from previous steps, since high-error regions in the next time step will coincide to a large degree with high-error regions in the previous time step. Thus, when a refined element's error falls below a user-defined percentage $H_{min} \in [0, 1)$ of the tolerance, we explicitly coarsen the element by deleting its underlying fine elements. Low-error elements are not coarsened, however, if a difference of more than one level of refinement at coarse/fine mesh interfaces would result from the deletion of the underlying fine elements.

We then predict the polynomial degrees needed for the next time step, with the restriction that an element's degree is increased by one only if it is in a smooth region. After a time step is accepted, if $E_{ij} > H_{max}\varepsilon/IJ$ for $H_{max} \in (0, 1]$, we increase the degree of $\mathbf{U}_{ij}(x, y, t + \Delta t)$ and define $\mathbf{U}_{ij}^{p+2}(x, y, t + \Delta t)$ as previously described. If $E_{ij} < H_{min}\varepsilon/IJ$ for $H_{min} \in [0, 1)$, we decrease the degree

of $\mathbf{U}_{ij}(x, y, t+\Delta t)$ and $\mathbf{U}_{ij}^{p+1}(x, y, t+\Delta t)$ by unity by setting the coefficients of the p- and $(p+1)$-degree terms, respectively, to zero.

Example 1. Consider the two-dimensional inviscid Burgers' equation

$$u_t + (\tfrac{1}{2}u^2)_x + (\tfrac{1}{2}u^2)_y = 0, -1 < x, y < 1, t > 0, \tag{16a}$$

with

$$u^0(x, y) = \tfrac{1}{2} + \tfrac{1}{2}\sin\pi(x+y), -1 \le x, y \le 1, \tag{16b}$$

and periodic boundary conditions. We solve (16) using adaptive hp-refinement with a local error tolerance 2.5×10^{-5} on a 16×16-element base mesh with $p = 0$ initially. In Figure 2, we show the exact solution (left) and the adaptive hp-refinement mesh (right) generated at three different times. The solution of (16) steepens and develops discontinuities at $t = 0.45$. At $t = 0.0625$ (20 base-mesh time steps), the solution is still smooth, and the adaptive hp-refinement method has performed only p-refinement (see the top of Figure 2) with polynomials of degrees $p = 0$, 1, and 2. At $t = 0.1875$ (60 base-mesh time steps), the solution is beginning to steepen along two fronts (see the middle of Figure 2), and the adaptive hp-refinement method has used one level of mesh refinement in the steep-solution regions, but has maintained high-order polynomials where the solution is smooth. At $t = 0.5$ (160 base-mesh time steps), shocks have developed (see the bottom of Figure 2), and the adaptive method has used two levels of mesh refinement near the shocks, while at most piecewise linear approximations satisfy the error tolerance in smooth-solution regions.

4. Dynamic Load Balancing via Tiling

Wheat's [24] tiling load-balancing system is a modification of the global load-balancing technique of Leiss and Reddy [17, 18] that is applicable to a wide class of two-dimensional, uniform-grid applications. Global balance is achieved by performing local balancing within overlapping processor neighborhoods, where each processor is defined to be the center of a neighborhood. Local balance involves element migrations to processors in the same neighborhood that have elements sharing edges.

Some modification of the original tiling system [24] is required by the adaptive hp-refinement algorithm. Because elemental work loads may vary due to p-refinement, the tiling algorithm must account for elemental work loads when performing local load balancing. In addition, because of the local

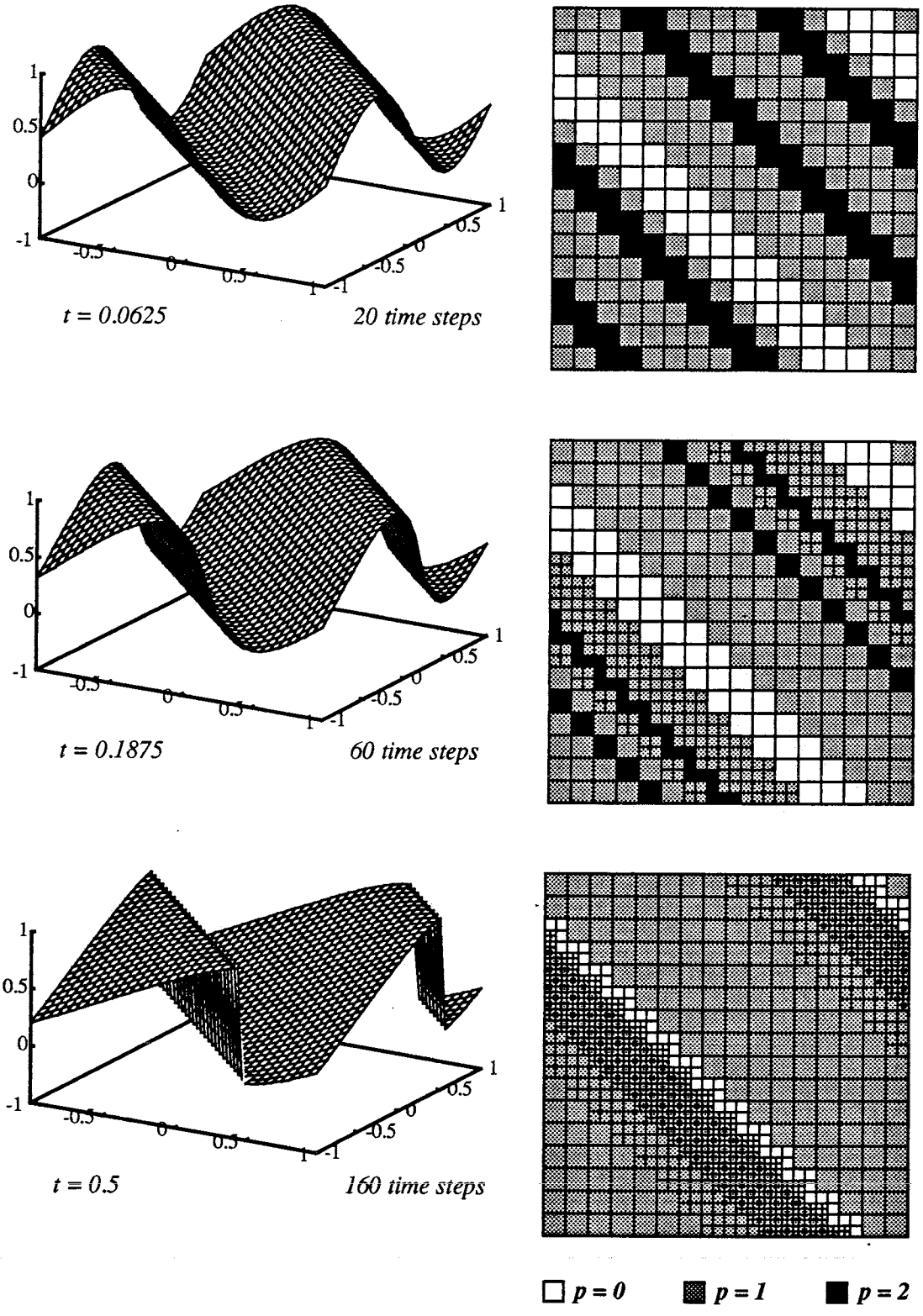


Figure 2. Exact solution (left) and adaptive hp-refinement mesh (right) for Example 1. The intensity of the shading (right) increases with the degree of the piecewise polynomials.

time-stepping strategy used for h-refinement and the synchronous nature of the explicit Runge-Kutta methods, all processors must compute on the same mesh level at the same time. Localized refinement, then, would cause processors in refined regions to be busy while processors without refinement are idle. Thus, load balancing must be applied individually at each mesh level. Refined elements may then be migrated to different processors from their coarse parent elements, resulting in additional communication when fine-mesh solutions are interpolated to the coarse mesh. The cost of this additional communication, however, is outweighed by the improved execution time resulting from better load balancing.

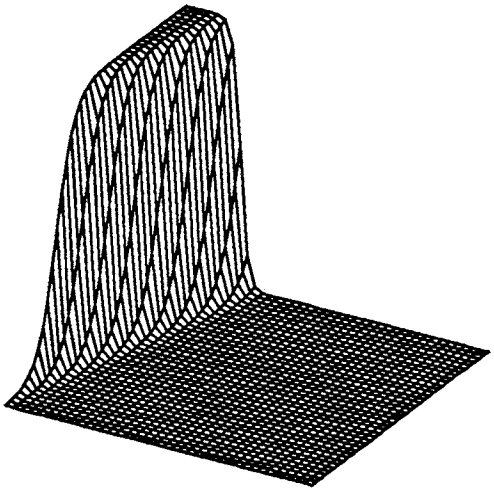
Example 2. In Figure 3, we show the influence of tiling on an adaptive hp-refinement computation that has been distributed over four processors. The solution is shown in the upper left of Figure 3. The adaptive hp-refinement algorithm uses piecewise polynomial approximations with $p = 0, 1$, and 2, and places one level of mesh refinement along the front, as shown in the upper right of Figure 3. If the original decomposition of the domain were uniform, only processors 0 and 2 would have any refined elements. However, tiling is applied to each mesh level to produce the resulting distribution shown at the bottom of Figure 3. On the refined mesh, elements are redistributed to all four processors. At each level, processors with high-order elements have far fewer elements than processors with low-order elements.

5. The Tiling Algorithm

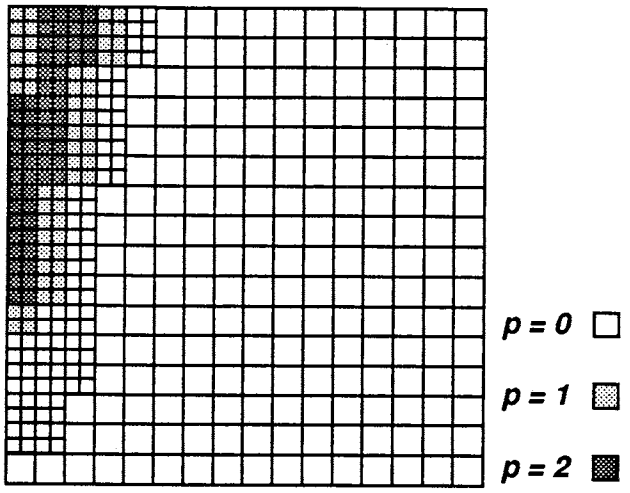
In tiling, a *processor neighborhood* includes a center processor and all other processors having elements that share edges with the center processor (see Figure 4). Processors that do not have any local elements on the mesh level being balanced use Leiss and Reddy's [17] definition of a neighborhood as processors within a given (unit) radius of the center processor.

Load balancing is performed after each local time step on each mesh level. Only one iteration of the tiling algorithm is performed; thus, the system is not iterated to global balance in a balancing phase.

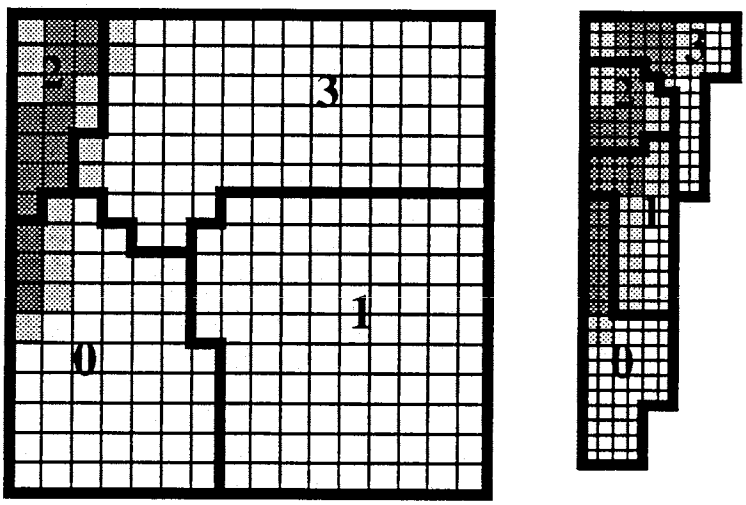
In the tiling algorithm, each processor determines its work load as the time to process its local data since the previous balancing phase less the time for any interprocessor communication performed during the computation phase. Neighborhood average work loads are also calculated. Each processor then compares its work load to the work load of the other processors in its neighborhood and determines which processors 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, but a single processor may receive several work requests.



Exact Solution



Adaptive *hp*-Refinement Mesh



Decomposition of Adaptive Mesh on 4 Processors

Figure 3. Adaptive hp-refinement (upper right) for the solution (upper left) of Example 2 and the decomposition generated by tiling on four processors (bottom).

Processors that received work requests prioritize them based on the request size. To prevent over-satisfaction of work requests when individual elements' processing costs vary widely due to p-refinement, we compute individual processing costs and use them to restrict the amount of actual work exported. Thus,

$$work\_available = proc\_work\_load - nborhd\_avg\_work\_load, \tag{17a}$$

and

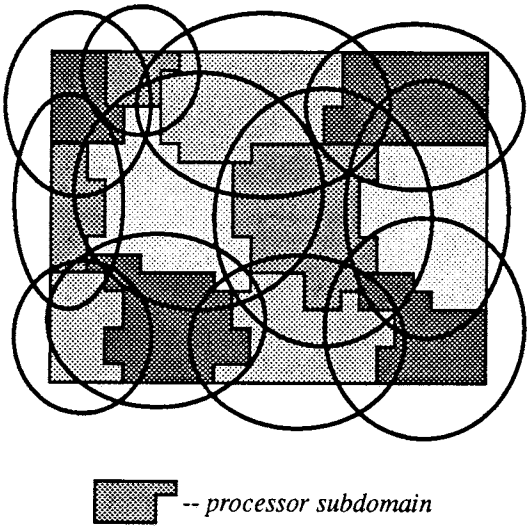


Figure 4. Example of 12 processors in 12 neighborhoods using the tiling definition of a neighborhood.

$$work\_migrated = \sum_{e \in selected\ elts} elemental\_cost_e \leq min(work\_available, work\_request)\ . \quad (17b)$$

In this way, a processor's work load cannot fall below its neighborhood average work load, preventing the development of oscillations of work between neighboring processors.

Each exporting processor selects a number of elements to migrate to satisfy (17). Elements along the boundary with the requesting processor are assigned priorities (initially zero) based upon the locality of their element neighbors. A priority scheme is used that emphasizes exporting elements that have neighbors on the same mesh level in the importing processor since the majority of communication is done within a level, but it attempts to reduce the number of elements migrated away from their parents so that inter-mesh communications are decreased. For each neighboring element on the same mesh level in the importing processor, the priority is increased by 3. For each neighboring element on the same mesh level in the exporting processor, the priority is decreased by 3. The priority is incremented for each coarse neighbor and parent element in the importing processor, and decremented for each coarse neighbor and parent element in the exporting processor. When an element has no neighboring elements in its local processor, it is advantageous to export it to any processor having its neighbors. Thus, "orphaned" elements are given the highest export priority. When two or more elements have the same priority, the processor selects the element with the largest work load that does not cause the exported work to exceed the work request or the work available for export (17). For exports to non-loaded processors, the exporting processor gives priority to elements closest to the neighboring processor; e.g., a request from a processor to the east would be

satisfied with the east-most elements in the exporting processor's subdomain. This technique for selecting elements results in a "peeling" of elements off the processor boundary, preventing the creation of "narrow, deep holes" in the element structures.

Once elements to be exported have been selected, the importing processors and those processors sharing edges with the migrating elements are notified. The neighboring processors update pointers in elements neighboring the migrating elements. Importing processors allocate space for the incoming elements, and the elements are transferred.

6. Data Structures and Interprocessor Communication

Elements are managed by data structures that maintain element connectivity and data position information, as shown in Figure 5. Since tiling is applied to each mesh level individually, these data structures are duplicated for each mesh level. Each tiling element contains pointers to its four neighboring elements, to its coarse parent element, and to its fine child elements.

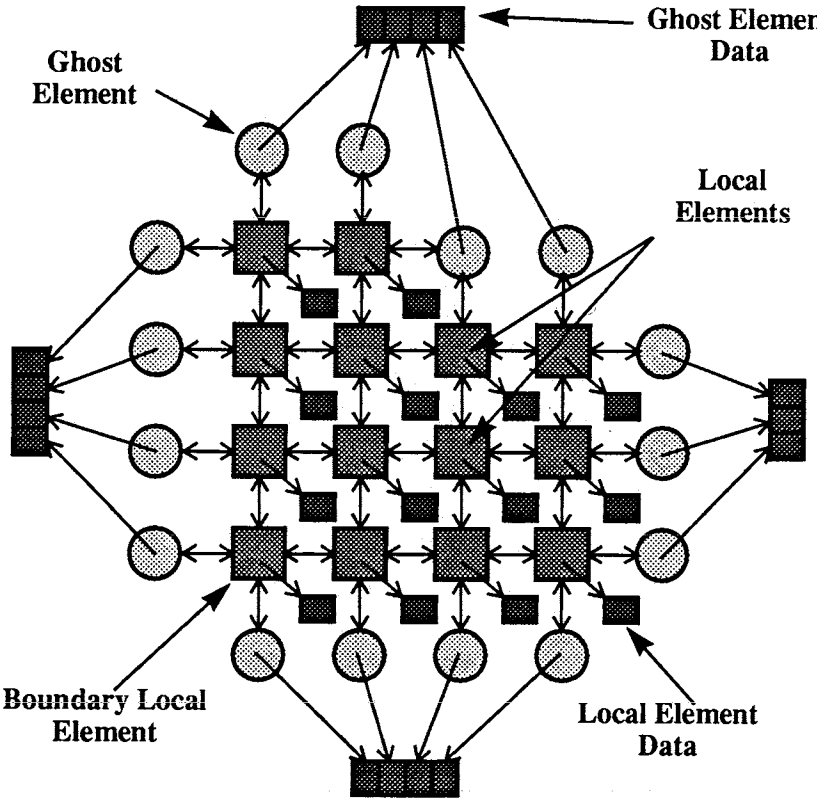


Figure 5. Element interconnection diagram [24].

Local elements are elements on which the processor performs the finite element computation. They are stored in a height-balanced, binary tree (AVL tree [2]) to allow efficient insertion and deletion during

migration. During the computation phase, local elements are accessed by an in-order traversal of this tree. Space for local element application data is allocated at the beginning of the computation and as needed for imported or fine-mesh elements.

On each mesh level, the ghost element tree contains the neighboring elements on the same mesh level that are local to other processors. The ghost element data of an individual mesh are communicated at each stage of the Runge-Kutta method on that mesh. Since the ghost elements needed for the finite element computation can change when elements are migrated, they are also stored in an AVL tree for easy insertion and deletion. Ghost-element data are stored contiguously, so a processor can receive the data in a single message from each neighbor and read the messages directly into the ghost element data space (i.e., without buffering the data and then moving it into the ghost element data space).

Data gather operations are needed to send boundary element data to neighboring processors. Thus, boundary elements are maintained in binary trees, one for each neighboring processor, facilitating proper ordering of the boundary element data during gather operations.

Elements along the edge of a refined mesh have neighboring elements in the coarse mesh. If the coarse neighbor is in the same processor as the fine element, the fine element simply sets its neighbor pointer to the coarse element. For coarse neighbors local to other processors, the fine mesh level maintains a separate tree to store the data associated with the coarse neighbor. The coarse mesh level also tracks those elements that are neighbors to fine mesh elements on other processors so that it can communicate the values to the fine mesh's processors at the end of the coarse mesh time step. These elements are maintained in trees analogous to the boundary element trees; however, since the coarse neighbor elements are not necessarily on the processor's subdomain boundary, constructing these trees is generally more costly than constructing the boundary element trees.

Resetting neighbor pointers after mesh refinement is complicated because child elements can migrate from their parents' processors. The child and parent pointers and element IDs stored in the local elements are used to reset the pointers. The element IDs of parent and child elements are communicated along processor subdomain boundaries. Newly created fine elements' neighbor pointers are set by following pointers through their parents (which are local) to their parents' neighbors (which may or may not be local) to the element IDs of the parents' neighbors' children. Neighbor pointers of old fine elements are reset after

additional refinement by following pointers through their coarse neighbors (which may or may not be local) to the element IDs of the coarse neighbors' children.

Communication between the fine and coarse mesh levels is needed to allow interpolation of the fine mesh values to the coarse mesh elements when parent and child elements are on different processors. Trees are constructed at both the coarse and fine mesh levels to mark those elements that will receive and send data, respectively, for interpolation. This communication is generally more expensive than the boundary exchanges and the coarse neighbor communications since many more element values must be communicated, but it is performed less frequently.

Communication is also needed from the coarse to the fine mesh levels. When elements of the fine mesh should be coalesced to decrease the amount of refinement in a particular region, parent elements must indicate this condition to their child elements. Flags in the coarse element tell the fine elements whether to coalesce. The flags are communicated to non-local child elements in the same way the fine-to-coarse-mesh communication is done.

7. Adaptive *hp*-Refinement Examples with Tiling

Example 3. We solve (16) on a 16×16-element base mesh with $p = 1$ initially using adaptive *hp*-refinement with a local error tolerance of 5.0×10^{-6} on 256 processors of the nCUBE/2. In Table 1, we examine the total execution time and the total maximum communication, computation, and load-balancing times of the method without balancing and with balancing performed once each local time step. The total maximum time is defined as

$$TotMax(time) = \sum_{i=1}^{\text{No. of time steps}} \max_{\text{all processors } P} (time \text{ used by processor } P \text{ in time step } i) \ . \qquad (18)$$

With balancing, the total maximum computation time (18) is reduced by 76.3% relative to the adaptive method without balancing. The total maximum communication time is increased 43.9% by balancing. The total maximum balancing overhead is 40.99 seconds. Despite the tiling overhead and additional communication time, however, the total execution time of the adaptive method is reduced 55.8% using tiling.

In Table 1, we also show the performance of the fixed-mesh, fixed-order method on a 112×112-element mesh with $p = 2$. The non-adaptive method attains high parallel efficiency and nearly-perfect load balance. However, the total execution time required to compute a solution with comparable accuracy is more than 4.5 times greater with the non-adaptive method than with the adaptive *hp*-refinement method and tiling.

	Adaptive *hp*-Refinement Method		Fixed-Mesh, Fixed-Order Method
	No Balancing	With Tiling	No Balancing
Global L^1-Error	0.0220026	0.0220026	0.0218864
TotMax(Computation Time)	2474.40 secs.	585.80 secs.	5291.71 secs.
Avg. / Max. Work Ratio	0.208	0.878	0.994
TotMax(Communication Time)	319.49 secs.	459.73 secs.	583.65 secs.
TotMax(Balancing Time)	0.0 secs.	40.99 secs.	0.0 secs.
Total Execution Time	2909.50 secs.	1285.78 secs.	5858.89 secs.

Table 1. Performance comparison for Example 3 using a fixed-mesh, fixed-order method on a 112x112-element mesh with p = 2 and the adaptive hp-refinement method without balancing and with balancing once for each local time step.

Example 4. We solve the two-dimensional Euler equations for a problem involving a Mach 10 shock in air $(\gamma = 1.4)$ moving down a channel containing a wedge oriented at a 30° angle to the channel [25]. The Euler equations have the form (1) with

$$\mathbf{u} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ e \end{bmatrix}, \qquad \mathbf{f} = \begin{bmatrix} \rho u \\ P + \rho u^2 \\ \rho uv \\ u(P+e) \end{bmatrix}, \quad \text{and} \quad \mathbf{g} = \begin{bmatrix} \rho v \\ \rho uv \\ P + \rho v^2 \\ v(P+e) \end{bmatrix}, \tag{19a}$$

where ρ, e, and P are the density, energy and pressure, and u and v are the velocity components in the x- and y-directions, respectively. The system is completed by the equation of state

$$P = (\gamma - 1)\left(e - \tfrac{1}{2}\rho\,(u^2 + v^2)\right). \tag{19b}$$

The eigenvectors and eigenvalues used in the projection of (19) to characteristic space are explicitly known [20].

Following Woodward and Colella [25], we solve the problem on a rectangular domain $-0.3 \le x \le 3.7$, $0 \le y \le 1$, with the wedge oriented so that it lies along the bottom boundary of the domain, $y = 0$, $1/6 \le x \le 3.7$. The initial conditions

$$\begin{bmatrix} \rho \\ u \\ v \\ P \end{bmatrix} = \begin{cases} [8.0, 4.125\sqrt{3}, -4.125, 116.5]^T, & \text{if } y \le \sqrt{3}\,(x - \frac{1}{6}), \\ [1.4, 0, 0, 1.0]^T, & \text{otherwise,} \end{cases} \tag{19c}$$

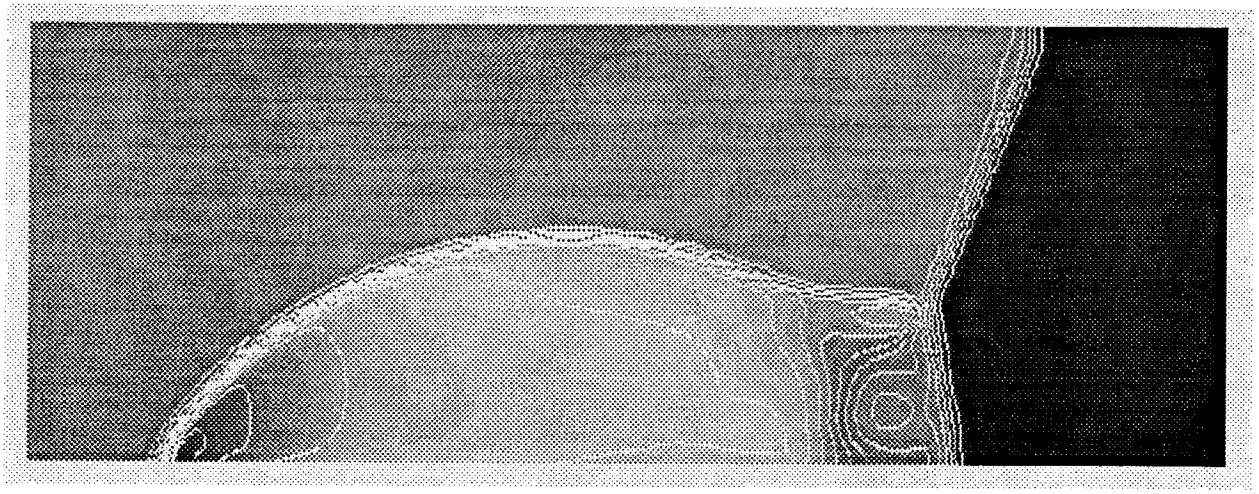
represent a Mach 10 shock making a 60° angle with the reflecting edge. Along the left boundary ($x = -0.3$) and the bottom boundary ($y = 0$) where $x \le 1/6$, Dirichlet boundary conditions representing the initial post-shock flow (19c) are prescribed. Values which describe the undisturbed motion of the initial Mach 10 shock are applied along the upper boundary ($y = 1$). Normal derivatives of the solution variables are set to zero along the right boundary ($x = 3.7$). Reflecting boundary conditions are used along the wedge ($y = 0$, $1/6 \le x \le 3.7$).

We solve (19) using the adaptive *hp*-refinement method with a 32×16-element base mesh and $p = 1$ initially. In Figure 6, we show the numerical solution for density (middle) and the adaptive *hp*-refinement mesh (bottom) at time $t = 0.2$. We also show a density profile (top) computed with a fixed-order, fixed-mesh method with $p = 2$ and 128×64-element mesh. The non-adaptive computation took 14,896 seconds on 128 processors of an Intel Paragon computer; the adaptive computation without load balancing required only 9411 seconds.

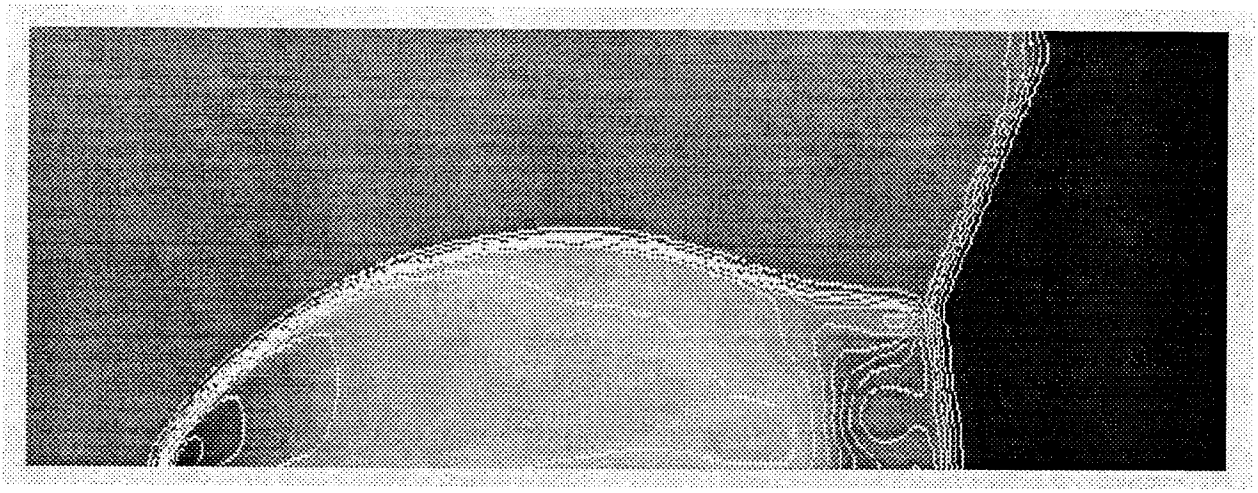
In Table 2, we show the performance of the tiling algorithm with the adaptive *hp*-refinement method for solving (19) on 128 processors of the Intel Paragon. Tiling was applied once on each mesh level for each local time step. With balancing, the total maximum computation time (18) is reduced 74.0% relative to the adaptive method without balancing. Average total communication time increases when load balancing is used; however, the total maximum communication time (18) decreases slightly for this example. The total execution time is reduced 66.9% by using tiling. The total maximum tiling overhead is 34.37 seconds, only 1.1% of the total execution time.

8. Conclusion

We have developed an adaptive parallel *hp*-refinement local finite element procedure for solving vector systems of conservation laws on rectangular domains. The solution strategy utilizes a local finite element procedure [9, 10] with solution limiting that preserves high-order accuracy near smooth extrema. Load



Fixed-mesh, fixed-order method (128x64-element mesh with p = 2)



Adaptive hp-refinement method (32x16-element base mesh with p = 1 initially)

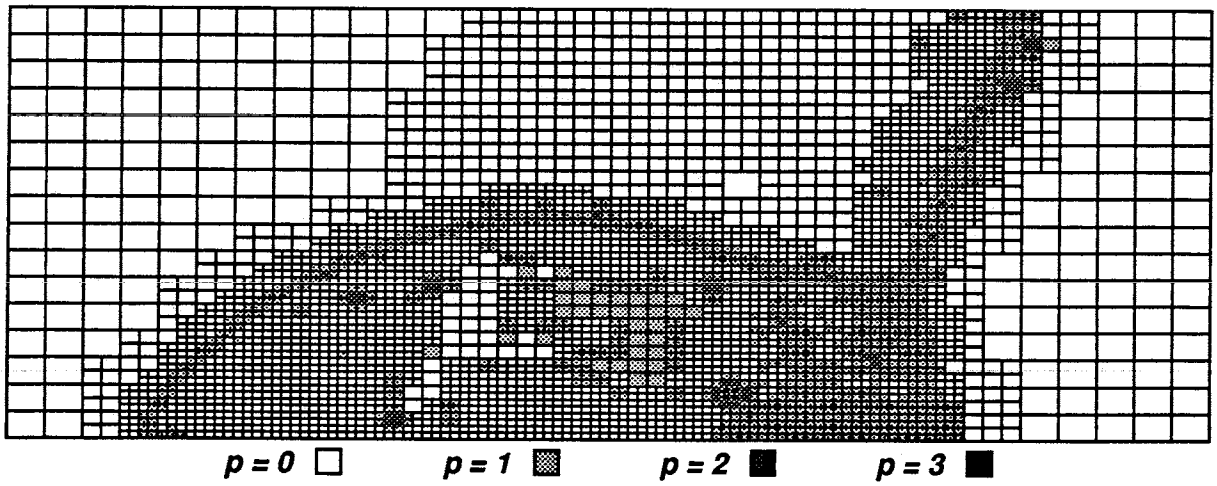


Figure 6. Density contours at t=0.2 for Example 4 using a fixed-mesh, fixed-order method (top) and adaptive hp-refinement (middle) with the orders and mesh shown at the bottom.

	Adaptive *hp*-Refinement Method		Fixed-Mesh, Fixed-Order Method
	No Balancing	With Tiling	No Balancing
TotMax(Computation Time)	4796.29 secs	1248.52 secs	14,500.63 secs.
Avg. / Max. Work Ratio	0.238	0.915	0.983
Avg. Total Communication Time	324.56 secs.	860.01 secs.	299.11 secs.
TotMax(Communication Time)	1202.76 secs.	1173.78 secs.	357.74 secs.
TotMax(Balancing Time)	0 secs.	34.37 secs.	0 secs.
Total Execution Time	9411.30 secs.	3112.79 secs.	14,896.38 secs.

Table 2. Performance comparison for Example 4 using a fixed-mesh, fixed-order method on a 128x64-element mesh with p = 2 and the adaptive hp-refinement method without balancing and with balancing once for each local time step.

balancing, performed by an extension of Wheat's [24] tiling procedure, appears to be efficient as it requires only 1-3% of the total solution time (Section 7). It is also effective by restoring the average-to-maximum processor work ratio to approximately 90% of ideal.

Efficient data structures used for adaptive refinement and tiling include trees of grids with finer grids regarded as offspring of coarser ones. Within each grid, AVL tree structures permit easy insertion and deletion of elements as they migrate between processors. Similar tree structures at inter-processor boundaries facilitate the transfer of data between neighboring processors.

Error estimates computed by *p*-refinement are robust and reliable but somewhat expensive. Simpler procedures using local computations [5, 12, 13] will provide improved performance, but their reliability must be examined. Additionally, Bey's procedure [5] is the only one known to apply on unstructured meshes.

The use of the limiting to indicate a preference for *h*- or *p*-refinement is effective in locating discontinuities and applying *h*-refinement there. However, when solving the Euler equations of Example 4, the procedure placed too great an emphasis on *h*-refinement (see Figure 6). Modification is necessary to ensure a more rapid transition from *h*- to *p*-refinement as the distance from a discontinuity increases.

Portions of this effort have been extended to three dimensions [7] and work in this direction will continue. Current methods on unstructured meshes of tetrahedral elements [7] use the local finite element method with piecewise constant approximations and adaptive *h*-refinement [11]. Future efforts will explore

extensions of the limiting procedures and data structures to unstructured-mesh environments with local time-stepping and order variation.

9. References

[1] S. Adjerid, J. E. Flaherty, P. K. Moore, and Y. Wang. "High-Order Adaptive Methods for Parabolic Systems." *Physica-D*, 60 (1992), 94-111.

[2] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, 1974.

[3] M. Berger and J.S. Saltzman, "AMR on the CM-2." *Appl. Numer. Maths.*, **14** (1994), 239-253.

[4] M.J. Berger and J. Oliger. "Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations." *J. Comp. Phys.*, **53** (1984), 484-512.

[5] K. Bey. *An hp-Adaptive Discontinuous Galerkin Method for Hyperbolic Conservation Laws*. Ph.D. Dissertation, University of Texas, Austin, 1994.

[6] R. Biswas, K. Devine, and J. Flaherty. "Parallel, Adaptive Finite Element Methods for Conservation Laws." *Appl. Numer. Maths.*, **14** (1994), 255-283.

[7] C.L. Bottasso, H.L. deCougny, M. Dindar, J.E. Flaherty, C. Ozturan, Z. Rusak, and M.S. Shephard, "Compressible Aerodynamics Using a Parallel Adaptive Time-Discontinuous Galerkin Least-Squares Finite Element Method." *AIAA* Paper 94-1888, *12th AIAA Appl. Aerodyn. Conf.*, June 20-22, 1994, Colorado Springs.

[8] K. Clark, J.E. Flaherty, and M.S. Shephard, Eds., *Appl. Numer. Maths.*, **14** (1994), special issue on Adaptive Methods for Partial Differential Equations.

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

[10] B. Cockburn 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.

[11] H.L. deCougny, K.D. Devine, J.E. Flaherty, R.M. Loy, C. Ozturan, and M.S. Shephard, "Load Balancing for the Parallel Adaptive Solution of Partial Differential Equations." *Appl. Numer. Maths.*, **16** (1994), 157-182.

[12] K. Devine. *An Adaptive hp-Refinement Finite Element Method with Dynamic Load Balancing for the Solution of Hyperbolic Conservation Laws on Massively Parallel Computers*. Ph.D. Dissertation, Rensselaer Polytechnic Institute, Troy, 1994.

[13] K. Devine, J. Flaherty, R. Loy, and S. Wheat. "Parallel Partitioning Strategies for the Adaptive Solution of Conservation Laws." *Proceedings of the Workshop on Modeling, Mesh Generation, and Adaptive Numerical Methods for Partial Differential Equations*, IMA, University of Minnesota, July, 1993.

[14] C.W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, 1971.

[15] Y. Kallinderis and A. Vidwans, "Generic Parallel Adaptive-Grid Navier-Stokes Algorithm." *AIAA J.*, **32** (1994), 54-61.

[16] F. Lafon and S. Osher. "High-Order Filtering Methods for Approximating Hyperbolic Systems of Conservation Laws." *ICASE* Report No. 90-25, ICASE, NASA Langley Res. Ctr., Hampton, 1990.

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

[18] H.N. Reddy. *On Load Balancing*. Ph.D. Dissertation, University of Houston, Houston, 1989.

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

[20] P.L. Roe. "Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes." *J. Comp. Phys.*, **43** (1981), 357-372.

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

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

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

[24] S. Wheat. *A Fine Grained Data Migration Approach to Application Load Balancing on MP MIMD Machines*. Ph.D. Dissertation, University of New Mexico, Albuquerque, 1992.

[25] P. Woodward and P. Colella. "The Numerical Simulation of Two-Dimensional Fluid Flow with Strong Shocks." *J. Comp. Phys.*, **54** (1984), 115-173.