

# Fast, Adaptively Refined Computational Elements in 3D

Craig C. Douglas<sup>1,2</sup>, Jonathan Hu<sup>3</sup>, Jaideep Ray<sup>3</sup>, Danny Thorne<sup>1</sup>, and Ray Tuminaro<sup>3</sup>

<sup>1</sup> University of Kentucky, Department of Computer Science, 325 McVey Hall,  
Lexington, KY 40506-0045, USA  
{douglas, thorne}@ccs.uky.edu

<sup>2</sup> Yale University, Department of Computer Science, P.O. Box 208285  
New Haven, CT 06520-8285, USA  
douglas-craig@cs.yale.edu

<sup>3</sup> Sandia National Laboratory, Livermore, CA 94550, USA  
{jhu, jairay, tuminaro}@california.sandia.gov

**Abstract.** We describe a multilevel adaptive grid refinement package designed to provide a high performance, serial or parallel patch class for use in PDE solvers. We provide a high level description algorithmically with mathematical motivation. The C++ code uses cache aware data structures and automatically load balances.

## 1 Introduction

In this paper, we use adaptively refined [2, 3, 15] multilevel [5, 11, 12] procedures to solve problems of the form

$$\begin{cases} \mathcal{L}(\phi) = \rho \text{ in } \Omega, \\ \mathcal{B}(\phi) = \gamma \text{ on } \partial\Omega, \end{cases} \quad (1)$$

subject to standard conditions that ensure ellipticity and well posedness [1]. The procedure is derived from the adaptive grid refinement process, not from the multigrid procedure.

This paper assumes the reader is familiar with how to discretize and solve a partial differential equation. For remedial information, see [8, 13, 16].

In §2, mathematical formalities are provided. In §3, we define a simple problem and then use it in later sections to motivate the definitions and methods. In §4, we define a multilevel adaptive mesh refinement method that is both complicated and directly implemented in C++ classes. In §5, we provide some implementation details. In §6, we draw some conclusions.

## 2 Mathematical Formalities

The basic algorithms are geometrically inspired. Definitions that assume nesting are defined from a domain (or subdomain), not a grid, perspective. This is utterly common in adaptive grid refinement literature, but is less so in multigrid literature.

We begin by assuming that  $\Omega$  is overlaid by a union of tensor product meshes  $A^{1,j}$ ,  $j = 1, \dots, n_1$ ,

$$G^1 = \bigcup_{j=1}^{n_1} A^{1,j}, \quad \text{where } G^1 \subset \Omega^1 = \Omega.$$

Normally  $n_1 = 1$ , but the method works fine for  $n_1 > 1$ . This is referred to as the level 1, or coarsest, grid and there will be operators defined on it later.

We may have many patches that have been determined through an adaptive grid refinement process. The set of local grid patches corresponding to  $\ell - 1$  refinements ( $1 < \ell \leq lmax$ ) are denoted

$$G^\ell = \bigcup_{j=1}^{n_\ell} A^{\ell,j}$$

where the  $A^{\ell,j}$  are also tensor product meshes and have been obtained by adaptively refining the  $A^{\ell-1,j}$  meshes. We define the domains  $\Omega^\ell$  and  $\Omega^{\ell,j}$  as the minimum domains that include  $G^\ell$  and  $A^{\ell,j}$ , respectively. Normally,  $\Omega^\ell$  will be a union of disconnected subdomains (one subdomain corresponding to each level  $\ell$  patch).

Note that since our code is really an adaptive grid refinement code with a multigrid procedure added as an afterthought,  $lmax$  can change (increase or decrease) during the course of solving an actual problem.

We assume there are projection and refinement operators defined,  $P$  and  $R$ , respectively. The notation is standard adaptive mesh refinement notation, but is different from multigrid notation (where the symbols are reversed, sadly). The operators are used interchangeably with either domains  $\Omega^\ell$  or grids  $G^\ell$ . In terms of superscripts of domains or grids,  $P$  *projects* from “fine to coarse,” i.e.,  $\ell \rightarrow \ell - 1$  and  $R$  *refines* from “coarse to fine,” i.e.,  $\ell \rightarrow \ell + 1$ .

We require strict nesting of grids from a geometric viewpoint:

$$G^{lmax} \subseteq G^{lmax-1} \subseteq \dots \subseteq \dots \subseteq G^1.$$

The domain version of this requirement can be written as

$$R(P(\Omega^{\ell+1})) \subseteq \Omega^{\ell+1} \quad \text{and} \quad P(\Omega^{\ell+1}) \subset \Omega^\ell$$

and

$$\Omega = \bigcup_{\ell=1}^{lmax} (\Omega^\ell - P(\Omega^{\ell+1})), \quad \text{where } \Omega^{lmax+1} = \phi.$$

Note that interpolation can be used to extend the method to nonnested grids quite easily, however.

The use of tensor product meshes allows for fairly straightforward finite difference/finite volume type stencils to define the discrete operator. At internal patch boundaries, however, some care must be taken. The general idea is to define

ghost points near internal patch boundaries so that locally equispaced unknowns are available for use with a regular stencil. In essence, simple B-splines [4] are used along the boundaries to produce the needed ghost point values. The key point is that the resulting discretization enforces  $C^1$  continuity along the patch boundaries.

$C^1$  continuity is different than what is normally required in the multigrid literature (which normally only imposes  $C^0$  continuity). For serial computing,  $C^0$  continuity is usually sufficient. However, for problems with severe fronts or near discontinuities in the solution,  $C^0$  continuity is not always sufficient or desirable.

For parallel multigrid with a local point relaxation smoother, only requiring  $C^0$  continuity requires a special procedure for data that is next to processor imposed subdomain boundaries. Otherwise the method is normally globally stable, but inconsistent. Hence the method does not necessarily converge through failing the well known numerical analysis theorem *stability+consistency=convergence*.  $C^1$  continuity imposes a process that guarantees consistency without disturbing stability. However, it imposes a  $C^1$  numerical solution, surprisingly, which turns out not to be much of a constraint for some very difficult problems.

The boundaries of the domains,  $\{\partial\Omega^\ell\}$ , are required to meet the following condition:

$$\partial\Omega^{\ell+1} \cap \partial\Omega^\ell \subset \partial\Omega$$

only. This merely ensures that the  $C^1$  continuity procedure that we use is well defined and of the right approximation order near patch boundaries in the interior of  $\Omega$  [11].

### 3 A Simple Example

For (1), assume we are solving Poisson's equation on the unit cube, with a cell centered finite volume method with uniform mesh spacing  $h_\ell$  on a level  $\ell$ . Note that our code supports variable coefficients as well, however.

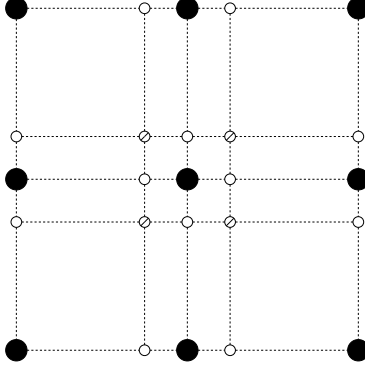
Let  $h_\ell = 2^{1-\ell}h_1$ . Let  $h = h_\ell$  and  $\phi = \phi^\ell$  unless otherwise noted, and let  $N = 1/h$  (the number of grid points in each dimension).

In the interior of  $\Omega^\ell$ , we have the standard seven point stencil:

$$(\Delta\phi)_{i,j,k} = (\phi_{i+1,j,k} + \phi_{i-1,j,k} + \phi_{i,j+1,k} + \phi_{i,j-1,k} + \phi_{i,j,k+1} + \phi_{i,j,k-1} - 6\phi_{i,j,k})h^{-2}.$$

On the physical boundaries, we must incorporate boundary values that match flux conditions necessary to ensure a  $C^1$  continuity. Assume the  $\phi$ 's with fractional indices are supplied boundary values. For instance, on the side boundary in the negative  $x$ -direction ( $i = 0$ ), we have

$$(\Delta\phi)_{0,j,k} = \left(\frac{4}{3}\phi_{1,j,k} + \frac{8}{3}\phi_{-\frac{1}{2},j,k} + \phi_{i,j+1,k} + \phi_{i,j-1,k} + \phi_{i,j,k+1} + \phi_{i,j,k-1} - 8\phi_{0,j,k}\right)h^{-2}.$$



**Fig. 1.** 3D Interpolation of Ghost Points, Step 1.

At the corner boundary where  $(i, j, k) = (0, N, N)$ , we have

$$(\Delta\phi)_{0,N,N} = \left( \frac{4}{3}\phi_{1,j,k} + \frac{8}{3}\phi_{-\frac{1}{2},j,k} + \frac{8}{3}\phi_{i,N+\frac{1}{2},k} + \frac{4}{3}\phi_{i,N-1,k} + \frac{8}{3}\phi_{i,j,N+\frac{1}{2}} + \frac{4}{3}\phi_{i,j,N-1} - 12\phi_{0,N,N} \right) h^{-2}.$$

The rest of the boundary discretizations can be produced similarly.

Near the interface between a coarse grid,  $\Omega^\ell$ , and a fine grid,  $\Omega^{\ell+1}$ , stencils must be defined on both the coarse and fine grid. The general idea is to use a flux differencing form of the equations  $\Delta\phi^\ell = \nabla \cdot f^\ell$ , where  $f^\ell = \nabla\phi^\ell$ :

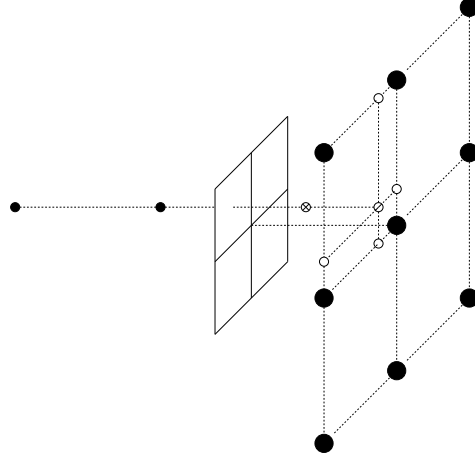
$$(\Delta\phi^\ell)_{i,j,k} = \frac{1}{h_\ell} \left( f_{i+\frac{1}{2},j,k}^\ell - f_{i-\frac{1}{2},j,k}^\ell + f_{i,j+\frac{1}{2},k}^\ell - f_{i,j-\frac{1}{2},k}^\ell + f_{i,j,k+\frac{1}{2}}^\ell - f_{i,j,k-\frac{1}{2}}^\ell \right).$$

Normally, these fluxes are defined as

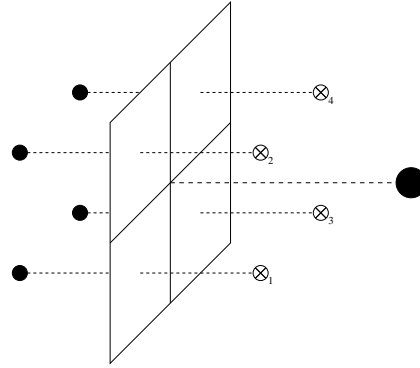
$$f_{i+\frac{1}{2},j,k}^\ell = (\phi_{i+1,j,k}^\ell - \phi_{i,j,k}^\ell) h_\ell^{-1}, \text{ and } f_{i-\frac{1}{2},j,k}^\ell = (\phi_{i,j,k}^\ell - \phi_{i-1,j,k}^\ell) h_\ell^{-1},$$

and similarly in the  $y$  and  $z$  directions. However, some of the  $\phi_{i,j,k}^\ell$ 's may not be available. To approximate a flux across a coarse edge that spans several fine grid edges (of a neighboring patch), each fine grid edge flux is first approximated using ghost values defined via interpolation or extrapolation. The individual fine grid fluxes are then summed to obtain the flux across the coarse edge. The interpolation/extrapolation procedure guarantees  $C^1$  continuity near a coarse-fine interface in the interior of  $\Omega$ .

As a concrete example, consider Figs. 1 and 2. Only one dimensional, quadratic interpolation and extrapolation is used. First (Fig. 1), the coarse element centers, represented by the large ●s, are used to calculate values at the ○ points, which are used in turn to calculate the values at the ⊙ points. Second (Fig. 2), we use these ⊙ points and two existing fine element centers, represented by the small ●s, to get ghost point values at the ⊗ points, which are at the centers of where fine elements would be.



**Fig. 2.** 3D Interpolation of Ghost Points, Step 2.



**Fig. 3.** 3D Flux Matching.

These  $\otimes$  points are used to approximate the average of the fluxes across the four fine grid cell walls at the coarse-fine interface. For instance,

$$f_{i-\frac{1}{2},j,k}^\ell = \frac{1}{4} \left( \frac{1}{h^{\ell+1}} (\delta_1 + \delta_2 + \delta_3 + \delta_4) \right),$$

where

$$\begin{aligned} \delta_1 &= u_{\otimes_1}^{\ell+1} - u_{2(i-1),2j-1,2k-1}^{\ell+1}, & \delta_2 &= u_{\otimes_2}^{\ell+1} - u_{2(i-1),2j,2k-1}^{\ell+1}, \\ \delta_3 &= u_{\otimes_3}^{\ell+1} - u_{2(i-1),2j-1,2k}^{\ell+1}, & \text{and } \delta_4 &= u_{\otimes_4}^{\ell+1} - u_{2(i-1),2j,2k}^{\ell+1}, \end{aligned}$$

as illustrated in Fig. 3, where the coarse-fine interface is in the negative  $x$ -direction from the coarse grid point at which the operator is being applied.

There can be up to three coarse-fine interfaces at a given coarse cell. In the case of multiple coarse-fine interfaces, an analogous flux matching procedure is

applied at each interface. A minor difference is that some of the  $\oslash$  points must be extrapolated from the coarse grid data.

More than three coarse-fine interfaces at one coarse cell is a case that is avoided by the mesh refinement algorithm. More generally, isolated coarse grid cells, squeezed between fine grid regions, is a case that is avoided by the mesh refinement algorithm. We must have three adjacent (and colinear) coarse grid cells, including the one at which the flux matching is being applied, in order to do the quadratic interpolation of ghost points. When coarse-fine interfaces abut the physical boundary, boundary values may be used in the interpolation procedure to compute the  $\oslash$  points.

## 4 Multilevel Adaptive Mesh Refinement

In this section, we define the operators, vectors, and algorithms needed to solve (1) numerically on a composite grid. Much of the material follows [11].

For solving (1) on level  $\ell$ , we need to store data and information about

$$G^\ell, \phi^\ell, \text{ and } \rho^\ell.$$

However, only data and information that cannot be associated with a finer level  $\ell + \epsilon$ ,  $\epsilon > 0$ , is stored on level  $\ell$ . Hence, only the part of level  $\ell$  associated with  $\Omega^\ell - P(\Omega^{\ell+1})$  is stored. Data is stored for grid points only on the finest grid that that a grid point exists. This is the opposite storage strategy from the one used for hierarchical basis multigrid implementations [18].

We define two discrete versions of  $\mathcal{L}$ : one that is defined on  $\Omega^\ell - P(\Omega^{\ell+1})$  and another that is defined assuming that no finer level exists. The difference is both subtle and confusing. We take great care to motivate the difference between the two and, in particular, how the patch boundary computations differ.

First, we define  $\mathcal{L}^\ell(\phi)$  on  $\Omega^\ell - P(\Omega^{\ell+1})$ . In the interior of  $\Omega^\ell$ , this is the standard discretization. By the  $\Omega^\ell/\Omega^{\ell+1}$  interface, we extrapolate ghost points, then do the standard discrete operator. We always use either the physical boundary conditions or the flux matching condition across the boundary.

Second, we define  $\mathcal{L}^{n_f, \ell}(\epsilon^\ell, \epsilon^{\ell-1})$  on  $\Omega^\ell$ . This is the standard discretization on  $G^\ell$  without any regard for the existence of finer levels. On  $G^1$ , we use the standard discretization across all of  $\Omega$ . On  $G^\ell$ ,  $\ell > 1$ , we only use coarse grid data to extrapolate ghost point information.

Multigrid methods always have at least one solver (called a smoother or rougher) and sometimes more than one of each. In the multilevel algorithm that we will define shortly, we need an operator  $S^\ell(\epsilon^\ell, R^\ell, h_\ell)$  on  $G^\ell$ .

The operator  $S^\ell$  is typically a damped Gauss-Seidel iteration using either the natural, red-black, or a multi-color ordering. First we compute  $S^\ell$  pointwise on level  $\ell$  without regard for any other level:

$$\epsilon_{ijk}^\ell \leftarrow \epsilon_{ijk}^\ell + \lambda [\mathcal{L}^{n_f, \ell}(\epsilon^\ell, 0) - R_{ijk}^\ell].$$

All coarse grid boundary components are set to zero above:  $\epsilon^{\ell-1} = 0$ , i.e., Dirichlet type boundary conditions are imposed. The damping factor for a Gauss-Seidel

operator  $S^\ell$  can be chosen to be

$$\lambda_{interior} = \frac{h_\ell^2}{6} \quad \text{and} \quad \lambda_{boundary} = \frac{5\lambda_{interior}}{6}.$$

These damping factors correspond to the reciprical of the discrete operator diagonal at the interior and boundary.

We need a method for computing a composite residual on coarser levels. On the finest level,

$$R^{lmax} = \rho^{lmax} - \mathcal{L}^{lmax}(\phi),$$

where  $\phi$  is defined over all composite grids. On any level  $\ell < lmax$ ,

$$R^\ell = \begin{cases} \text{Average}(R^{\ell+1} - \mathcal{L}^{nf,\ell+1}(\epsilon^{\ell+1}, \epsilon^\ell)) & \text{in } P(\Omega^{\ell+1}) \\ \rho^\ell - \mathcal{L}^\ell(\phi) & \text{in } \Omega^\ell - P(\Omega^{\ell+1}) \end{cases}$$

where Average() is a standard nine point weighted restriction from the finer level to the coarser level [5].

Lastly, we define a composite grid version of a multigrid  $\mu$  cycle, where  $\mu$  determines how many correction cycles to do from a given level  $\ell > 1$ .

```

Algorithm MGCycle( $\ell, \mu$ )
  If ( $\ell = lmax$ ) then  $R^\ell \leftarrow \phi^\ell - \mathcal{L}^{nf,\ell}(\phi^\ell, \phi^{\ell-1})$ 
  If ( $\ell = 1$ ) then
    solve  $\mathcal{L}^1 e^1 = R^1$  on  $G^1$ 
  Otherwise repeat  $\mu$  times :
     $\phi^{\ell,save} \leftarrow \phi^\ell$ 
     $e^\ell, e^{\ell-1} \leftarrow 0$ 
     $e^\ell \leftarrow S^\ell(e^\ell, R^\ell, h^\ell)$ 
     $\phi^\ell \leftarrow e^\ell$ 
     $R^{\ell-1} \leftarrow \text{Average}(R^\ell - \mathcal{L}^{nf,\ell}(e^\ell, e^{\ell-1}))$  on  $P(\Omega^\ell)$ 
    MGCycle( $\ell - 1, \mu$ )
     $e^\ell \leftarrow e^\ell + \text{Interpolate}(e^{\ell-1})$ 
     $R^\ell \leftarrow R^\ell - \mathcal{L}^{nf,\ell}(e^\ell, e^{\ell-1})$ 
     $\bar{e}^\ell \leftarrow 0$      $\bar{e}^\ell \leftarrow S^\ell(\bar{e}^\ell, R^\ell, h^\ell)$ 
     $e^\ell \leftarrow e^\ell + \bar{e}^\ell$ 
     $\phi^\ell \leftarrow \phi^{\ell,save} + e^\ell$ 

```

Choosing  $\mu = 1$  is the multigrid V cycle. Choosing  $\mu = 2$  is the multigrid W cycle. If we double the number of iterations of the smoother or rougher each time we move to a coarser grid, we get a variable V or W cycle.

## 5 Implementation Details

We have implemented a set of C++ classes to solve elliptic PDEs. Our main usage is in solving nonlinear elliptic and parabolic PDEs for complex problems, e.g., combustion simulation.

To implement multigrid on adaptively refined meshes, we use two main types of objects: grids and grid functions. In this section we describe our grids and grid functions and methods that are defined for each.

The classes store data and operate on it based on the following: discrete operators, solvers, residuals, and a composite multilevel algorithm. The form implemented follows the descriptions in §4.

A grid is, first of all, a bounding box. It contains the beginning and ending coordinates of a brick shaped grid stored as integer coordinates of the beginning and ending grid points from the discretization. We also store the real coordinates of the corresponding region from the domain. In addition, a grid includes the mesh spacing in each direction, the number of grid points in each direction, and the identity of the processor that it belongs to.

Multiple grids are combined to form a grid level. A grid level is characterized by its mesh spacing, and the mesh spacings between adjacent levels differ by a factor, usually two or four, called the refinement factor. Multiple grid levels are combined to form a grid hierarchy.

Finer grids can be thought of as either nested within or hovering above a coarser grid. In either case, they can be called child grids of the coarser grid. Grids maintain pointers to their children and parent. A grid has at most one parent.

In the beginning, a grid hierarchy usually consists of only one grid level with only one grid. Through the refinement and load balancing processes this may develop into an elaborate hierarchy of many grid levels with many grids designed to efficiently resolve the features of the problem.

Grid functions store the values associated with each grid point on a grid. A grid function is initialized with a reference to the corresponding grid. The grid functions of all the grids in the hierarchy are combined into a composite grid function for which methods are written for applying operations over all the grid functions of the grid hierarchy as a whole. All the grid functions exist on all the processors, but they only allocate space for data on the processor that their corresponding grid belongs to.

We derive these grid functions from an array class that has been highly optimized for cache performance and robustness. Grid function data can be accessed using Fortran-like syntax that allows for memory management to be done at a very low level. This allows us to experiment with different cache optimizations on different processors easily.

We are exploring a number of storage schemes for optimizing cache effects. For serial computers, modifying algorithms and data structures for standard multigrid has been investigated for a variety of problems [7, 9, 10, 17]. For adaptively refined grids on parallel processors, there are many more possibilities than in the serial case. We are implementing many possibilities in order to see which ones work well on which parallel architectures.

After approximating the solution on the grid hierarchy, there may be points at which the solution is not sufficiently accurate according to some error approximation such as Richardson extrapolation [11]. Clustering is the process of

grouping these points into conveniently sized regions. We use clustering routines based on ones in GrACE (Grid Adaptive Computation Engine) [14], which is an object-oriented C++ package for managing adaptively refined structured meshes. It provides an infrastructure for storage of and computation on adaptively refined meshes.

Bigger regions are more convenient for doing computations. There is overhead associated with applying operations to each grid, so minimizing the number of grids by making the grids bigger means less overhead. Also, more work has to be done at the interface between a coarse and fine grid, so it is better to use bigger grids which have a greater proportion of interior nodes.

On the other hand, smaller regions are more convenient for load balancing and getting an entire patch to fit into cache. It is hard to distribute a few big grids evenly over many processors. Smaller regions are desirable since they can more efficiently capture the groups of points that need refinement. Since the refinement regions are brick shaped and the groups of points needing refinement are most likely not, it is probable that the refinement region will include points that do not need refinement. Using smaller grids makes it possible to minimize the number of points included that do not need refinement.

Refinement is the process of building finer grids over the regions computed by the clustering process. For each such region, a new grid is created and incorporated into the hierarchy. The mesh spacing for the new grid is equal to the mesh spacing for its parent grid divided by the refinement factor. The new grid creates a pointer to its parent grid, and the parent grid creates a pointer to its new child grid.

Load balancing is the process of determining a distribution of the grids over more than one processor in a way that will balance the amount of work each processor has to do. We use Zoltan [6] for load balancing.

The load balancing procedure assigns each grid to a processor. The grid functions must then distribute their data accordingly. They might need to move their data to a different processor and they might need to interpolate their data to the grid function of a new grid that was created in the refinement stage. Grid functions and composite grid functions are equipped with methods for handling both of these tasks: message passing and interpolation.

The components of the solution procedure are the  $\mathcal{L}$  operators, the smoother, and the transfer operators, projection and interpolation. and implement what was described in §4.

We provide the user with an object that is composed of the grid hierarchy object and the composite grid function object. The code uses methods for initializing the base grid and grid function(s) and then uses a solve method. We try to minimize the programming required of the user while guaranteeing robustness.

## 6 Conclusions

There are a number of strategies for implementing adaptively refined mesh solvers for elliptic problems in 3D. Besides what type of meshes are chosen,

there are nonobvious caching techniques that must be implemented and evaluated in as portable a manner as possible. By portable, a small set of easily determined parameters must be left to the user to choose, preferably in as automatic a manner as possible. The library described in this paper is providing a useful testbed for such an evaluation, besides being useful software.

## References

1. AGMON, S.: *Lectures on Elliptic Boundary Value Problems*. Van Nostrand Reinhold, New York, 1965.
2. BERGER, M.J., AND COLELLA, P.: Local adaptive mesh refinement for shock hydrodynamics. *J. Comput. Phys.* 82 (1989), 64–84.
3. BERGER, M.J., AND OLIGER, J.: An adaptive mesh refinement for hyperbolic partial differential equations. *J. Comput. Phys.* 53 (1984), 484–512.
4. BOOR, C.: *A Practical Guide to Splines*. Springer-Verlag, New York, 1978.
5. BRIGGS, W.L., HENSON, V.E., AND MCCORMICK, S.F.: *A Multigrid Tutorial*. SIAM Books, Philadelphia, 2000. Second edition.
6. DEVINE, K., HENDRICKSON, B., BOMAN, E., ST. JOHN, M., AND VAUGHAN, C.: Design of dynamic load-balancing tools for parallel applications. In *Proc. International Conference on Supercomputing* (Santa Fe, 2000).
7. DOUGLAS, C.C.: Caching in with multigrid algorithms: problems in two dimensions. *Paral. Alg. Appl.* 9 (1996), 195–204.
8. DOUGLAS, C.C., HAASE, G., AND LANGER, U.: A tutorial on elliptic pde's and parallel solution methods. <http://www.mgnet.org/~douglas/ccd-preprints.html>, 2002.
9. DOUGLAS, C.C., HU, J., KOWARSCHIK, M., RÜDE, U., AND WEISS, C.: Cache optimization for structured and unstructured grid multigrid. *Elect. Trans. Numer. Anal.* 10 (2000), 21–40.
10. HU, J.: *Cache Based Multigrid on Unstructured Grids in Two and Three Dimensions*. PhD thesis, University of Kentucky, Department of Mathematics, Lexington, KY, 2000.
11. MARTIN, D., AND CARTWRIGHT, K.: Solving Poisson's equation using adaptive mesh refinement. <http://seesar.lbl.gov/anag/staff/martin/tar/AMR.ps>, 1996.
12. MCCORMICK, S.F.: The fast adaptive composite (FAC) method for elliptic equations. *Math. Comp.* 46 (1986), 439–456.
13. MORTON, K.W., AND MAYERS, D.F.: *Numerical Solution of Partial Differential Equations*. Cambridge University Press, Cambridge, 1994.
14. PARASHAR, M.: GrACE. <http://www.caip.rutgers.edu/~parashar/TASSL/Projects/GrACE>, 2001.
15. RÜDE, U.: *Mathematical and Computational Techniques for Multilevel Adaptive Methods*, vol. 13 of *Frontiers in Applied Mathematics*. SIAM, Philadelphia, 1993.
16. VARGA, R.S.: *Matrix Iterative Analysis*. Prentice-Hall, Englewood Cliffs, NJ, 1962.
17. WEISS ET AL, C.: Dimepack. <http://www.bode.cs.tum.edu/Par/arch/cache>.
18. YSERENTANT, H.: On the multi-level splitting of finite element spaces. *Numer. Math.* 49 (1986), 379–412.