

## CACHE AWARE MULTIGRID ON ADAPTIVELY REFINED MESHES

Craig C. Douglas\*, Dan T. Thorne<sup>+</sup>, Jonathan Hu, Jaideep Ray, and Ray S. Tuminaro<sup>†</sup>

\*Department of Computer Science and Mechanical Engineering  
Center for Computational Sciences, University of Kentucky and  
Department of Computer Science, Yale University  
email: craig.douglas@yale.edu, URL: <http://www.mgnet.ord/~douglas>

<sup>+</sup>Earth Sciences Department, Florida International University  
email: thorned@fiu.edu

<sup>†</sup>Sandia National Laboratories (California)  
email: {jhu,jairay,tuminaro}@ca.sandia.gov

**Key words:** HPC, PDEs, solvers, iterative methods,  $C^1$ -interpolation

**Abstract.** *Many current computer designs, including the node architecture of most parallel supercomputers, employ caches and a hierarchical memory architecture. Therefore the speed of a code (e.g., multigrid) depends increasingly on how well the cache structure is exploited. The number of cache misses provides a better measure for comparing algorithms than the number of multiplies. Unfortunately, estimating cache misses is difficult to model a priori and only somewhat easier to do a posteriori.*

*Typical multigrid applications are running on data sets much too large to fit into the caches, even when adaptively refined meshes using a hierarchical multigrid method is used. Thus, copies of the data that are once brought into the cache should be reused as often as possible. For multigrid, the possible number of reuses is always at least as great as the number of iterations of the smoother or rougher (plus the residual correction before correction steps).*

*We develop and investigate a complicated adaptive mesh refined multilevel algorithm from a standard algorithm and then drastically simplify it.*

## 1 INTRODUCTION

Patch-based adaptive mesh refinement [2, 3] has become an attractive technique for spatial discretization in a host of scientific simulations, ranging from shock hydrodynamics [12] to combustion [5] and plasma physics [15]. The ability to place a refined patch (with a Cartesian mesh) at any arbitrary location in the domain obviates the need for mesh stretching and promotes its use with higher-order stencils [13]. Conformal transformation techniques have extended this technique to nonrectangular/cuboid domains of sufficient complexity to meet the needs of scientific (as opposed to engineering) simulations [8].

A large number of scientific problems (e.g. combustion, incompressible hydrodynamics, magnetohydrodynamics etc) frequently pose a Poisson problem as a part of their solution strategy. While this could be formulated as a giant  $\mathbf{A}\vec{\mathbf{x}} = \vec{\mathbf{b}}$  problem and treated with iterative methods (e.g. Krylov methods), the multilevel nature of the grid strongly suggests multigrid approaches. Further, the (relatively) small data set associated with a patch lends this approach to cache-based optimization of numerical operations local to a patch i.e. smoothing. In this paper, we use a combination of adaptive refinement [2, 3, 14] and multilevel [4, 9, 10] procedures to solve elliptic boundary value 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 solution procedure is derived from the adaptive mesh refinement process, not from the multigrid procedure. Hence, we use notation that is common in the adaptive mesh refinement (AMR) community.

This research focuses on

1. Implementing cache aware optimizations to multigrid in an AMR context,
2. Modifying a well known AMR multigrid algorithm [9] to do only post-smoothing so that the cache aware optimizations will have a greater effect.

Cache aware algorithms are designed to minimize the number of times data goes through cache, thereby increasing the efficiency of the algorithm. Cache memories are much faster than main memory, so the CPU can be kept more busy when it is getting data from cache memories. In an AMR context, we modify Gauss-Seidel so that all the data required by the smoothing iterations needs to be brought into a cache only once, not many times, and it still gets the exact same answer as the standard algorithm (i.e. bitwise compatibility). In order to further improve the efficiency of cache usage, we do only post-smoothing and combine the residual computation with the smoother. In this way all of the smoothing *and* the residual computation for each level of a V-Cycle can be accomplished while bringing the data into cache just once.

Doing post-smoothing only is a substantial change over the AMR algorithm used in [9]. It is especially useful when implementing cache aware algorithms as discussed in §5.

We see far better cache effects when more smoothing iterations are done consecutively. Further, reducing the work that is done outside of the smoother means that speeding up the smoother will have a greater impact on the whole algorithm.

This paper assumes the reader is familiar with discretization and numerical solution of partial differential equations [6, 11, 17].

In §2, we provide a mathematical background. In §3, we present a 2D example. In §4, the multilevel adaptive mesh refinement method is described. In §5, we discuss cache aware Gauss-Seidel and the V-Cycle. In §6, we present numerical results.

Many very highly technical intermediate details have been left out of this article. The omitted material is in [16].

## 2 MATHEMATICAL BACKGROUND

The basic algorithms are geometrically inspired. Definitions are based on a domain (or subdomain) rather than a grid perspective. This is standard in adaptive grid refinement literature but less standard in multigrid literature.

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

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

Normally  $n_1 = 1$ . However, the method works for  $n_1 > 1$ , too. This is referred to as the level 1, or coarsest grid. Operators are defined on it later.

An adaptive mesh refinement procedure is used to define many *patches*. The set of local grid patches corresponding to  $\ell - 1$  refinements ( $1 < \ell \leq \ell_{max}$ ) is denoted

$$\Lambda^\ell = \bigcup_{j=1}^{n_\ell} \Lambda^{\ell,j} \quad \text{and} \quad \Lambda^{\ell_{max}+1} = \emptyset.$$

The  $\Lambda^{\ell,j}$  are tensor product meshes that have been obtained by adaptively refining the  $\Lambda^{\ell-1,j}$  meshes. The definition for  $\Lambda^{\ell_{max}+1}$  is a convenience that simplifies a number of the algorithms we define throughout this paper. We define the domains  $\Omega^\ell$  and  $\Omega^{\ell,j}$  as the minimum domains that include  $\Lambda^\ell$  and  $\Lambda^{\ell,j}$ , respectively. Normally,  $\Omega^\ell$  is a union of disconnected subdomains (one subdomain corresponding to each level  $\ell$  patch). Note that within an adaptive grid refinement code  $\ell_{max}$  can change (increase or decrease) during the course of solving an actual problem.

The AMR procedure defines a composite grid,  $\Lambda_c^{\ell_{max}}$ , and more generally, a composite grid hierarchy,  $1 < \ell \leq \ell_{max}$ , by

$$\Lambda_c^\ell = \bigcup_{i=1}^{\ell} (\Lambda^i - \mathcal{P}(\Lambda^{i+1})), \tag{2}$$

where  $\mathcal{P}$  is the projection operator discussed below. The  $\ell^{th}$  composite grid,  $\Lambda_c^\ell$ , contains all points from the  $\ell^{th}$  level patches,  $\Lambda^\ell$ , as well as additional points from regions not covered by the patches. The new grid points correspond to mesh points from patches on lower levels, always taking from patches on the closest possible level.

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

We require nesting of domains

$$\Omega^{\ell_{max}} \subseteq \Omega^{\ell_{max}-1} \subseteq \dots \subseteq \dots \Omega^1 \equiv \Omega,$$

which can be written as

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

and

$$\Omega = \bigcup_{\ell=1}^{\ell_{max}} (\Omega^\ell - \mathcal{P}(\Omega^{\ell+1})), \quad \text{where } \Omega^{\ell_{max}+1} = \emptyset.$$

The use of tensor product meshes allows for fairly straightforward finite difference and finite volume stencils to define the discrete operator. At internal patch boundaries, however, some care must be taken. We define *ghost points* near internal patch boundaries and use quadratic interpolation to acquire values at these points so that locally equispaced unknowns are available for use with regular stencils within most of the computations. When computing composite grid residuals, however, more complicated stencils are needed for coarse points adjacent to finer grid patches. This is formally covered in §3. The main idea is to use the same interpolated values for the stencils on both the fine grid side and the coarse grid side when updating points that are adjacent to a coarse-fine interface. Hence, the fluxes used to approximate the operator across the coarse-fine interfaces are matched and continuity of the first derivative (i.e.  $C^1$  continuity) is enforced. This is referred to as *flux matching*. The  $C^1$  continuity at the boundary between the coarse and fine subdomains can be precisely defined in terms of the derivatives of the quadratic functions that interpolate the ghost points. See §3 for details of the ghost point interpolation procedure. The key point is that enforcing  $C^1$  continuity preserves the second order convergence of the method. Especially for problems with severe fronts or near discontinuities,  $C^0$  continuity alone is not always sufficient. The boundaries of the domains,  $\{\partial\Omega^\ell\}$ , are required to meet the condition

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

which ensures that the flux matching procedure that we use is well defined and of the right approximation order near patch boundaries in the interior of  $\Omega$  [9].

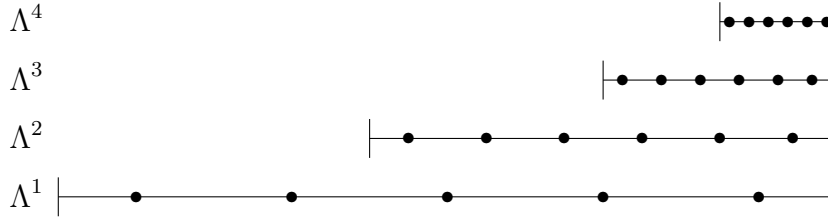


Figure 1: A sample grid hierarchy.

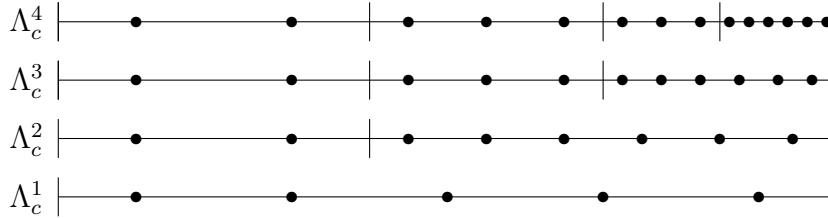


Figure 2: Composite grid hierarchy corresponding to the sample grid in Figure 1.

We approximate the solution to (1) using a multigrid algorithm to numerically solve the finite dimensional problem

$$\mathcal{L}_c^{\ell_{max}} \phi_c^{\ell_{max}} = \rho_c^{\ell_{max}}, \tag{3}$$

where  $\mathcal{L}_c^{\ell_{max}}$  is a matrix representing the discretization on  $\Lambda_c^{\ell_{max}}$ ,  $\phi_c^{\ell_{max}}$  are the unknowns, and  $\rho_c^{\ell_{max}}$  is the right hand side. Conceptually, the grid hierarchy used within the multigrid procedure is the composite grid hierarchy defined above. In practice, we design the implementation to be patch based (see §4). A simple 1D AMR patch hierarchy is illustrated in Figure 1. In the figure, the dots represent grid points and the vertical lines represent either a physical boundary or a patch boundary. A patch is a maximal set of contiguous grid points on a given level of the grid hierarchy. There can be multiple patches per level, although the grid hierarchy in Fig. 1 has only one patch per level. Figure 2 illustrates the corresponding composite grid hierarchy for the simple 1D example. Each level in this composite grid hierarchy is a composite grid defined in (2). Equation (3) is defined on the highest level composite grid, namely  $\Lambda_c^{\ell_{max}}$ . Operators  $\mathcal{L}_c^\ell$  are matrices representing the discretizations on composite grids  $\Lambda_c^\ell$  for all  $\ell$ . In §4, we define patch based versions of the discrete operator.

### 3 An Example

This section describes the tools that are needed to define patch based versions of the discrete operators used by the multilevel method presented in §4. The specific form of equation (1) that this discussion will address is

$$\begin{cases} -\nabla \cdot (a \nabla u) = \rho, & \text{in } \Omega, \\ u = \gamma, & \text{on } \partial\Omega, \end{cases}$$

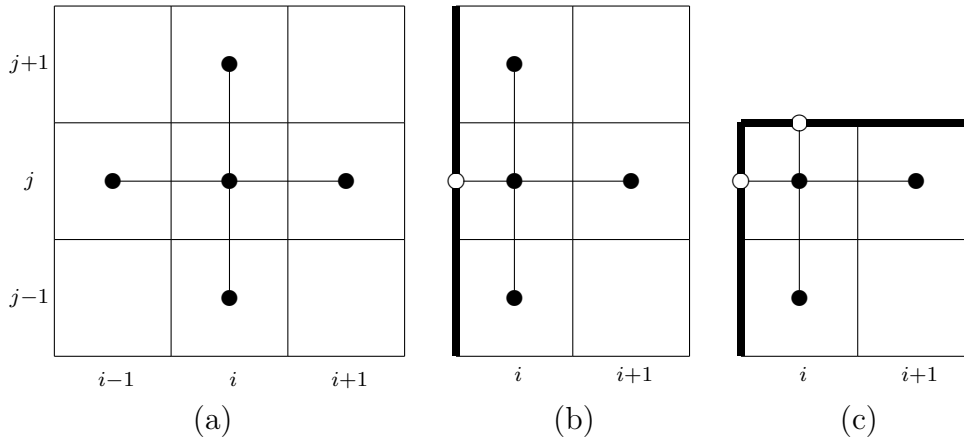


Figure 3: Stencils. The dark lines in (b) and (c) represent boundaries.

where  $\Omega$  is a rectangular domain and  $\gamma$  is a constant. In addition, this example will assume isotropic mesh spacing  $h = \Delta x = \Delta y$ .

Fig. 3 illustrates the stencils for an interior cell, an edge cell, and a corner cell. The formula for interior cells is the usual, the same as for vertex centered grids. For the edge stencil Fig. 3(b), Taylor’s series expansions around  $u_i$  for  $u_{i+1}$ ,  $u_{i-\frac{1}{2}}$ ,  $u_{j+1}$ , and  $u_{j-1}$  gives the discretization

$$(\Delta u)_{ij} \approx u_i'' + u_j'' \approx \left( \frac{4}{3}u_{i+1,j} + \frac{8}{3}u_{i-\frac{1}{2},j} + u_{i,j+1} + u_{i,j-1} - 6u_{ij} \right) h^{-2}.$$

For the corner stencil Fig. 3(c), Taylor’s series expansions around  $u_i$  for  $u_{i+1}$ ,  $u_{i-\frac{1}{2}}$ ,  $u_{j+\frac{1}{2}}$ , and  $u_{j-1}$  gives the discretization

$$(\Delta u)_{ij} \approx u_i'' + u_j'' \approx \left( \frac{4}{3}u_{i+1,j} + \frac{8}{3}u_{i-\frac{1}{2},j} + \frac{8}{3}u_{i,j+\frac{1}{2}} + \frac{4}{3}u_{i,j-1} - 8u_{ij} \right) h^{-2}.$$

In order to apply the discrete operator at interfaces between coarse and fine grids, *ghost points* are interpolated around patches. These ghost points are used to complete the stencils on the fine grid side of interfaces. A *flux matching* procedure, employing the ghost points, is used to define the operator on the coarse grid side of an interface. The ghost point interpolation and flux matching procedures are described in the following subsections.

### 3.1 Interpolation of ghost points in 2D

In order to apply regular stencils at the boundary points of patches and enforce flux matching at the coarse-fine interfaces, we need to interpolate values at ghost points around the edges of the patches. This section explains how to do these interpolations. They are necessary for both of the patch based operators  $\mathcal{L}^\ell$  and  $\mathcal{L}^{nf,\ell}$  defined in §4.

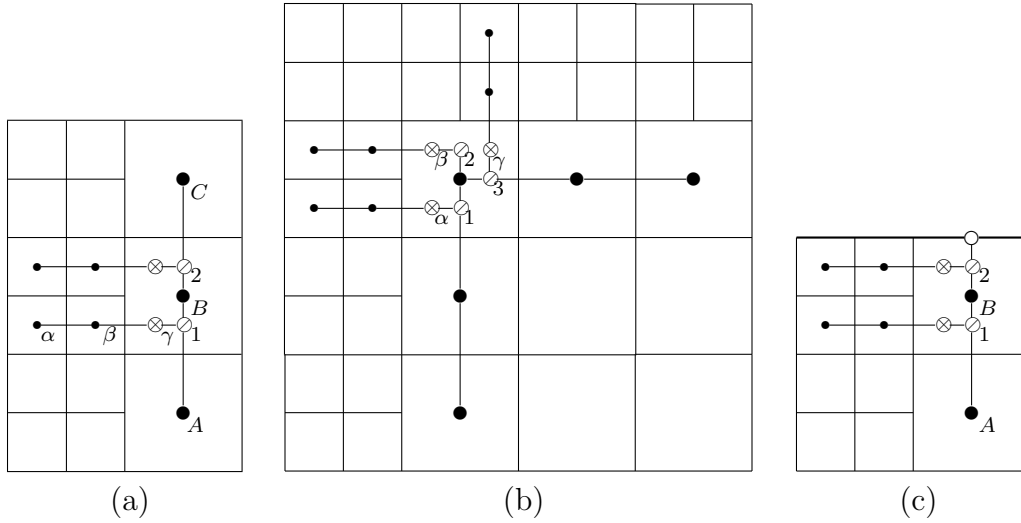


Figure 4: Interpolation of Ghost Points. The dark edge in (c) indicates a boundary.

### 3.1.1 One Coarse-Fine Interface

Values are needed at the  $\otimes$  ghost points for the one-sided coarse-fine interface illustrated in Figure 4(a). There are two steps. *Step 1:* Compute values for the  $\circ$  points. Let  $a = u(\bullet_A)$ ,  $b = u(\bullet_B)$  and  $c = u(\bullet_C)$ , and let  $h$  be the mesh spacing on the finer grid. We derive a  $C^1$  quadratic interpolation formula using

$$u(x) = c_0 + c_1x + c_2x^2,$$

where  $c_0 = a$ ,  $c_1 = \frac{1}{h}(b - \frac{1}{4}c - \frac{3}{4}a)$ , and  $c_2 = \frac{1}{h^2}(\frac{1}{8}c - \frac{1}{4}b + \frac{1}{8}a)$ . Then

$$u(\circ_1) = c_0 + c_1(\frac{3}{2}h) + c_2(\frac{3}{2}h)^2, \text{ and } u(\circ_2) = c_0 + c_1(\frac{5}{2}h) + c_2(\frac{5}{2}h)^2.$$

*Step 2:* Compute values for the  $\otimes$  points. Let  $a = u(\bullet_\alpha)$ ,  $b = u(\bullet_\beta)$  and  $c = u(\circ_1)$ , and let  $h$  be the mesh spacing on the finer grid. Again, we derive a  $C^1$  quadratic interpolation formula using

$$u(x) = c_0 + c_1x + c_2x^2,$$

where  $c_0 = a$ ,  $c_1 = -\frac{1}{15h}(21a - 25b + 4c)$ , and  $c_2 = \frac{2}{15h^2}(3a - 5b + 2c)$ . Then

$$u(\otimes_\gamma) = c_0 + c_1(2h) + c_2(2h)^2.$$

The procedure is analogous for the other  $\otimes$  point.

### 3.1.2 Two Coarse-Fine Interfaces

Ghost point interpolation at a two-sided coarse-fine interface is illustrated in Figure 4(b). Again, values are needed at the  $\otimes$  points, and there are two steps. The notation

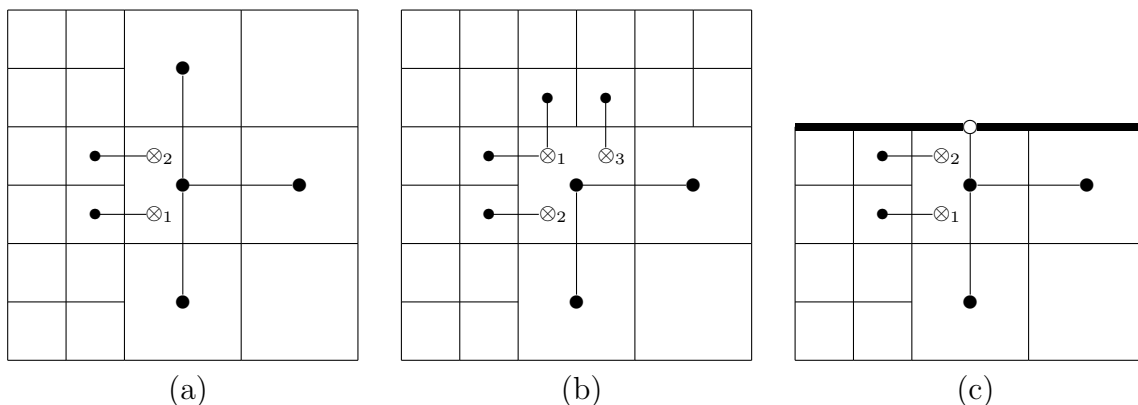


Figure 5: Flux matching for the variable coefficient case. The dark edge in (c) represents a boundary.

$c_{\{0,1,2\}}$  refers generically to the coefficients associated with each interpolation. The values of the coefficients are *not* the same for every interpolation.

*Step 1:* Use the same system as in *Step 1* of §3.1.1, once in each direction, to get the coefficients  $c_{\{0,1,2\}}$  needed to compute values for the  $\circlearrowleft$  points. *Step 2:* Use the same system as in *Step 2* of §3.1.1, twice in the  $x$ -direction and once in the  $y$ -direction, to get the coefficients  $c_{\{0,1,2\}}$  needed to compute values for the  $\otimes$  points.

### 3.1.3 At a Boundary

Ghost point interpolation at a coarse-fine interface adjacent to a boundary is illustrated in Figure 4(c). Once again, values are needed at the  $\otimes$  points, and there are two steps.

*Step 1:* Compute values for the  $\circlearrowleft$  points. Let  $a = u(\bullet_A)$ ,  $b = u(\bullet_B)$  and  $c = u(\circ)$ .

*Step 2:* Compute values for the  $\otimes$  points by the same system as in *Step 2* of §3.1.1.

## 3.2 Flux matching in 2D

This section introduces the flux-matching computations. Flux matching is used to avoid one-sided derivatives and preserve  $C^1$  continuity. It is necessary for the patch based operator  $\mathcal{L}^\ell$  used in §4.

The general form of the discretization is

$$-(f_n - f_s + f_e - f_w) = \rho_o.$$

The fluxes  $f_{\{n,s,e,w\}}$  for the different cases are given in the following subsections (assuming isotropic mesh spacing  $h = \Delta x = \Delta y$ ).

The computations in this section involve values interpolated at the ghost points  $\otimes$  from §3.1. The value of the solution at points  $\otimes_{\{1,2,3\}}$  is represented by  $u_{\otimes_{\{1,2,3\}}}^{\ell+1}$ .

### 3.2.1 One Coarse-Fine Interface

Flux matching at a one-sided coarse-fine interface is illustrated in Figure 5(a). The fluxes for this case are

$$\begin{aligned} f_{i+\frac{1}{2},j}^\ell &= \frac{1}{h_\ell} (u_{i+1,j}^\ell - u_{i,j}^\ell), \\ f_{i-\frac{1}{2},j}^\ell &= \frac{1}{2} \left( \frac{1}{h_{\ell+1}} (u_{\otimes_1}^{\ell+1} - u_{2(i-1),2j-1}^{\ell+1}) + \frac{1}{h_{\ell+1}} (u_{\otimes_2}^{\ell+1} - u_{2(i-1),2j}^{\ell+1}) \right) \\ f_{i,j+\frac{1}{2}}^\ell &= \frac{1}{h_\ell} (u_{i,j+1}^\ell - u_{i,j}^\ell), \\ f_{i,j-\frac{1}{2}}^\ell &= \frac{1}{h_\ell} (u_{i,j}^\ell - u_{i,j-1}^\ell). \end{aligned}$$

### 3.2.2 Two Coarse-Fine Interfaces

Flux matching at a two-sided coarse-fine interface is illustrated in Figure 5(b). The fluxes are the same as in 3.2.1 except for the north flux

$$f_{i,j+\frac{1}{2}}^\ell = \frac{1}{2} \left( \frac{1}{h_{\ell+1}} (u_{2i-1,2(j+1)-1}^{\ell+1} - u_{\otimes_2}^{\ell+1}) + \frac{1}{h_{\ell+1}} (u_{2i,2(j+1)-1}^{\ell+1} - u_{\otimes_3}^{\ell+1}) \right).$$

### 3.2.3 At a Boundary

Flux matching at a coarse-fine interface that is adjacent to a boundary is illustrated in Figure 5(c). The fluxes in the  $x$ -direction are the same as in 3.2.1. The fluxes in the  $y$ -direction are given by

$$f_n - f_s = \frac{1}{h_\ell} \left( \frac{8}{3} u_{i,j+\frac{1}{2}} + \frac{4}{3} u_{i,j-1} - 4u_{ij} \right).$$

## 4 Adaptive Mesh Refinement Multilevel Method

The AMR multigrid algorithm is shown in Alg. 1. The patch based discrete operators  $\mathcal{L}^\ell$  and  $\mathcal{L}^{nf,\ell}$  are restrictions of the composite grid operator onto the patches on level  $\ell$  with the interfaces between coarse and fine patches handled by ghost points and flux matching as shown in §3. The operator  $\mathcal{L}^\ell$  operates on  $\Lambda^\ell - \mathcal{P}(\Lambda^{\ell+1})$  using ghost points to complete the stencil at the exterior boundary of the patch and using flux matching to update the points at interior boundaries where there are finer grid patches. The operator  $\mathcal{L}^{nf,\ell}$  operates on entire patches  $\Lambda^\ell$  ignoring finer levels and, hence, does not require the flux matching procedure.

### 4.1 Post-smoothing only

Alg. 1 can be sped up considerably by only doing post-smoothing, as illustrated in Alg. 2. There are several advantages to this approach:

---

**Algorithm 1** AMR Multigrid V-Cycle.

---

**MG**(  $\ell, e^\ell, r^\ell, \rho^\ell, \phi^{\ell_{max}}$  )

- 1: **if**  $\ell == 1$  **then**
  - 2:  $e^\ell \leftarrow S^\ell(e^\ell, r^\ell)$  on  $\Lambda^\ell$
  - 3:  $\phi^{\ell_{max}} \leftarrow \phi^{\ell_{max}} + e^\ell$  on  $\Lambda^\ell - \mathcal{P}(\Lambda^{\ell+1})$
  - 4: **return**
  - 5: **end if**
  - 6: **if**  $\ell == \ell_{max}$  **then**
  - 7:  $r^\ell = \rho^\ell - \mathcal{L}^{nf,\ell}(\phi^\ell, \phi^{\ell-1})$
  - 8: **end if**
  - 9:  $e^\ell \leftarrow 0$ ;  $e^{\ell-1} \leftarrow 0$
  - 10:  $e^\ell \leftarrow S^\ell(e^\ell, r^\ell)$  on  $\Lambda^\ell$
  - 11:  $\phi^{\ell,temp} \leftarrow e^\ell + \phi^{\ell_{max}}$  on  $\Lambda^\ell - \mathcal{P}(\Lambda^{\ell+1})$
  - 12:  $\hat{r}^\ell \leftarrow r^\ell - \mathcal{L}^{nf,\ell}(e^\ell, e^{\ell-1})$  on  $\Lambda^\ell$
  - 13:  $r^{\ell-1} \leftarrow \mathcal{P}^\ell \hat{r}^\ell$  on  $\mathcal{P}(\Lambda^\ell)$
  - 14:  $r^{\ell-1} \leftarrow \rho^{\ell-1} - \mathcal{L}^{\ell-1}(\phi^{\ell,temp}, \phi^{\ell-1}, \phi^{\ell-2})$  on  $\Lambda^{\ell-1} - \mathcal{P}(\Lambda^\ell)$
  - 15: **MG**(  $\ell - 1, e^{\ell-1}, r^{\ell-1}, \rho^{\ell-1}, \phi^{\ell_{max}}$  )
  - 16:  $e^\ell \leftarrow e^\ell + \mathcal{R}^{\ell-1} e^{\ell-1}$
  - 17:  $r^\ell \leftarrow r^\ell - \mathcal{L}^{nf,\ell}(e^\ell, e^{\ell-1})$  on  $\Lambda^\ell$
  - 18:  $\bar{e}^\ell \leftarrow 0$
  - 19:  $\bar{e}^\ell \leftarrow S^\ell(\bar{e}^\ell, r^\ell)$  on  $\Lambda^\ell$
  - 20:  $e^\ell \leftarrow e^\ell + \bar{e}^\ell$  on  $\Lambda^\ell$
  - 21:  $\phi^{\ell_{max}} \leftarrow \phi^{\ell_{max}} + e^\ell$  on  $\Lambda^\ell - \mathcal{P}(\Lambda^{\ell+1})$
-

---

**Algorithm 2** Multigrid V-Cycle with post-smoothing only

---

**MG** ( $\ell, e^\ell, r^\ell, \rho^\ell, \phi^{\ell_{max}}$  )

- 1: **if**  $\ell == \ell_{max}$  **then**
  - 2:    $r^\ell = \rho^\ell - \mathcal{L}^{nf,\ell}(\phi^\ell, \phi^{\ell-1})$
  - 3: **end if**
  - 4: **if**  $\ell == 1$  **then**
  - 5:    $e^1 \leftarrow 0$
  - 6:   Solve  $\mathcal{L}_c^1 e^1 = r^1$  on  $\Lambda^1$
  - 7: **else**
  - 8:    $r^{\ell-1} \leftarrow \mathcal{P}^\ell r^\ell$  on  $\mathcal{P}(\Lambda^\ell)$
  - 9:    $r^{\ell-1} \leftarrow \rho^{\ell-1} - \mathcal{L}^{\ell-1}(\phi^\ell, \phi^{\ell-1}, \phi^{\ell-2})$  on  $\Lambda^{\ell-1} - \mathcal{P}(\Lambda^\ell)$
  - 10:   MG ( $\ell - 1, e^{\ell-1}, r^{\ell-1}, \rho^{\ell-1}, \phi^{\ell_{max}}$  )
  - 11:    $e^\ell \leftarrow \mathcal{R}^\ell e^{\ell-1}$  {Includes interpolation of ghost points.}
  - 12:    $e^\ell \leftarrow S^\ell(e^\ell, r^\ell)$  on  $\Lambda^\ell$
  - 13: **end if**
  - 14:  $\phi^{\ell_{max}} \leftarrow \phi^{\ell_{max}} + e^\ell$  on  $\Lambda^\ell - \mathcal{P}(\Lambda^{\ell+1})$
- 

1. The initial guess only needs to be set (to 0) on the base grid.
2. There is no need for a temporary correction  $\phi^{\ell,temp}$  on the projection side of the V-Cycle.
3.  $\hat{r}$  does not need to be computed.
4. The residual is only computed on one side of the V-Cycle.
5. There is no correction after interpolation and no need for an intermediate correction step before updating  $\phi^{\ell_{max}}$ .
6. The residual is not updated on the interpolation side of the V-Cycle.

In addition, merging the smoothing steps on each level onto one side of the V-Cycle leads to better speedups from the cache optimizations discussed in the next section.

## 5 Cache Optimizations

There are two stages to the cache optimizations. The first stage involves only the smoother. The smoothing updates are blocked and staggered so that the answer is the same as the standard application of the smoother but all the data only has to be fetched into cache

once. This is called a *cache aware smoother*. The second stage involves combining three of the AMR multigrid components: smoother, ghost point computation, and residual computation. They are combined in a cache aware manner like the cache aware smoother but with the operations from the three components interleaved. This is called *cache aware multigrid*.

### 5.1 Cache aware Gauss-Seidel

Consider naturally ordered Gauss-Seidel restricted to matrices  $A_j$  which are based on discretization methods which are local to only 3 neighboring rows of the grid. Partition the grid into blocks of  $\ell$  rows, and let  $m$  be the number of smoothing iterations required. It is necessary that  $\ell + m - 1$  rows of an  $N \times N$  grid  $G$  fit entirely into cache simultaneously and that  $m < \ell$ .

There are two special cases to the cache aware algorithm: the first block of rows and the rest of the blocks.

The first case is for the first  $\ell$  rows of the grid. The data associated with rows 1 to  $\ell$  is brought into cache. The data in rows 1 to  $\ell - m + 1$  are updated  $m$  times, and the data in rows  $j$ ,  $\ell - m + 2 \leq j \leq \ell$ , are updated  $\ell - j + 1$  times.

The second case is for the rest of the blocks of  $\ell$  rows of the grid. Once the first block of grid rows is partially updated, we have a second block to update and must also finish updating the first block of grid rows. After the  $i^{\text{th}}$  update in the second block, we can go back and update rows  $\ell$  down to  $\ell - i + 1$  in the first block of rows, always performing the updates in the order that preserves the dependencies in the standard iteration (so that the cache aware iteration will achieve bitwise the same answer as the standard iteration). This procedure is repeated in the remaining blocks of rows until all the blocks are updated. In effect, this is a domain decomposition methodology applied to the standard iteration. The result is that  $m$  updates are done while bringing all the data through cache only one time.

### 5.2 Cache aware multigrid

Integrating the residual computation with the cache-aware smoother can give better cache-effects in multigrid. Call this the *combined smoother*. The combined smoother is implemented for the post-smoothing only version of the algorithm.

The combined smoother computes the residual to be used in the next V-Cycle. It can only compute the part of the residual that is not covered by finer patches. On the projection side of the V-Cycle, the projection of the part of the residual that is from finer patches must now include the application of the flux matching procedure, because information needed for the flux matching is not available when the combined smoother is called.

A complication to interleaving the residual computation with the cache aware smoother is that, in the V-Cycle, the residual is updated *and* concurrently used as the right hand

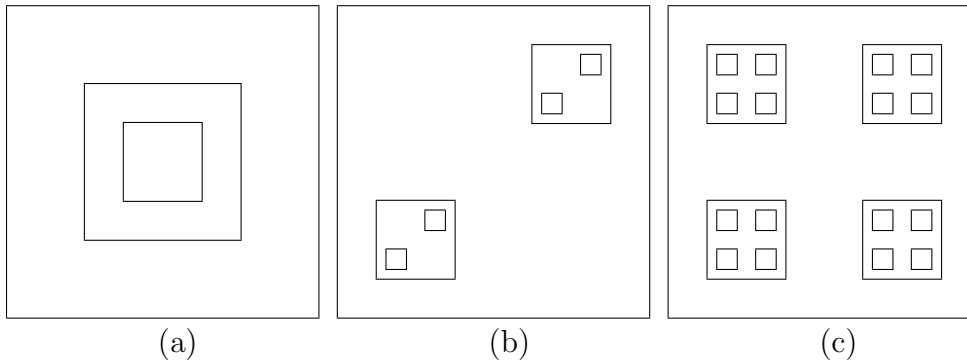


Figure 6: Refinement patterns: (a) One refinement per patch. (b) Two refinements per patch. (c) Four refinements per patch.

side for the Gauss-Seidel updates. So the residual updates need to be postponed long enough to avoid changing the right hand side for a subsequent Gauss-Seidel update. In addition, residual updates at the patch boundaries require updated ghost points around the patch, so ghost point interpolation must also be interleaved with the cache aware smoother.

## 6 NUMERICAL RESULTS

This section shows results on the hierarchies illustrated in Fig. 6 and also on a full domain refinement hierarchy (i.e. regular, non-AMR multigrid). The refinement patterns are not meaningful to the nature of the problem being solved here. They are contrived merely to demonstrate the behavior of the algorithm.

For the adaptive refinement cases, we do full domain refinements from a coarse grid for a few levels before applying the adaptive refinement. The coarse grid is  $8 \times 32$ , in our example. The first adaptive refinement is on level 6 which is a  $512 \times 2048$  grid in our example. Another way of explaining this is that we do adaptive refinement on a  $512 \times 2048$  base grid and then use geometric multigrid as the solver on that base grid.

The base grid discretizes the unit square  $[0, 1]^2$ . We solve the Poisson equation with the right hand side chosen so that the solution is

$$u(x, y) = \sin(\pi x) \sin(\pi y/4) x e^{x^2 + (y/4)^2}.$$

The initial guess on the base grid is  $u = 0$ , and we iterate until the composite residual is smaller than  $10^{-6}$  times the norm of the composite right hand side:

$$\|r_c\| < 10^{-6} \|\rho_c\|.$$

The tables show results for the AMR multilevel method employing the cache aware smoother, labeled *CA*, and the combined smoother, labeled *CAMG*, (as described in §5) compared with a standard implementation of the smoother. Speedups are based on

	IA1	IA2	PIII	PIV
Full	2.1824	2.1442	2.8288	2.3216
One	2.1038	2.1615	2.6543	1.9278
Two	2.2170	2.2448	2.7539	2.0469
Four	2.0226	2.0887	2.5493	1.7878

Table 1: Standard(2,2) Versus CAMG(0,4)

	CA	CAMG
Full	1.1554	1.3350
One	1.1032	1.4807
Two	1.0981	1.5501
Four	1.0807	1.4226

Table 2: Itanium 1, Standard(0,4) Versus CA(0,4) and CAMG(0,4)

elapsed wall clock time. The timings include only the solution procedure. They do not include initialization of the hierarchy and right hand sides.

The main speedups we are interested in are shown in Table 1. This table shows the speedups of the original standard AMR multigrid V-Cycle, Standard(2,2), compared to the cache aware post-smoothing only version, CAMG(0,4). We see speedups consistently over a factor of 2 and even close to a factor of 3 in some cases.

The contribution from just the cache optimizations alone is shown for the Itanium 1, Itanium 2, Pentium III and Pentium IV in Tables 2, 3, 4 and 5. These experiments show the speedups associated with the cache optimizations for the V(0,4) cycle. The CA(0,4) cycle uses just the cache aware smoother plugged into the multigrid algorithm. Those speedups are fairly insignificant in most cases. The CAMG(0,4) cycle shows the speedups associated with interleaving the residual computation with the smoother in a cache aware manner. These speedups are more substantial, up to nearly a factor of 2.

The contribution from merging the smoothing steps on each level that is accomplished by the post-smoothing only algorithm is shown in Table 6. This shows the speedup associated with doing post-smoothing only, CA(0,4), versus doing pre-smoothing *and* post-smoothing, CA(2,2). Note that we do the same total number of smoothing iterations per level in both cases, except for the projection side of the first V-Cycle and the correction side of the last V-Cycle. In particular, the pre-/post-smoothing case uses 2 smoothing iterations on each side. The post-smoothing only case uses 4 smoothing iterations only on the correction side of the V-Cycle. These examples were run with the cache aware smoothers. The average time per V-Cycle is about 1.4 to 2 times faster in the post-smoothing only runs.

	CA	CAMG
Full	1.1123	1.2754
One	1.0788	1.4564
Two	1.0766	1.5346
Four	1.0578	1.4246

Table 3: Itanium 2, Standard(0,4) Versus CA(0,4) and CAMG(0,4)

	CA	CAMG
Full	1.3574	1.6344
One	1.2399	1.7575
Two	1.2221	1.8417
Four	1.2031	1.7002

Table 4: Pentium III, Standard(0,4) Versus CA(0,4) and CAMG(0,4)

	CA	CAMG
Full	1.1432	1.3355
One	1.1310	1.3907
Two	1.1040	1.4356
Four	1.0849	1.2824

Table 5: Pentium IV, Standard(0,4) Versus CA(0,4) and CAMG(0,4)

	IA1	IA2	PIII	PIV
Full	1.7606	1.7797	2.0714	1.8841
One	1.4810	1.5388	1.6885	1.4752
Two	1.4933	1.5111	1.6513	1.4808
Four	1.4715	1.4956	1.6367	1.4409

Table 6: CA(2,2) Versus CA(0,4)

## 7 CONCLUSIONS

We experimented with cache aware smoothers and integration of the residual computation with the smoother. Both of these give good speedups. The integrated residual computation is especially useful for getting remarkable speedups in the AMR context.

Modifying the algorithm to do post-smoothing only is key to realizing good performance in the AMR context. It takes better advantage of the cache aware smoother since the smoothing iterations on each level are all contiguous.

Our future plans for this research are to extend the code to three dimensions with variable coefficients and support for parallelism. Progress has already been made on all of these tasks. In addition, we are going to experiment with a variety of other cache aware approaches. The 3D case, in particular, needs more sophisticated approaches, such as those discussed in [7].

## REFERENCES

- [1] S. AGMON, *Lectures on Elliptic Boundary Value Problems*, Van Nostrand Reinhold, New York, 1965.
- [2] M. J. BERGER AND P. COLELLA, *Local adaptive mesh refinement for shock hydrodynamics*, J. Comput. Phys., 82 (1989), pp. 64–84.
- [3] M. J. BERGER AND J. OLIGER, *An adaptive mesh refinement for hyperbolic partial differential equations*, J. Comput. Phys., 53 (1984), pp. 484–512.
- [4] W. L. BRIGGS, V. E. HENSON, AND S. F. MCCORMICK, *A Multigrid Tutorial*, SIAM Books, Philadelphia, 2000. Second edition.
- [5] M. S. DAY AND J. B. BELL, *Numerical Simulation of Laminar Reacting Flows with Complex Chemistry*, Combust. Theory Modelling, 4 (2000), pp. 535–556.
- [6] C. C. DOUGLAS, G. HAASE, AND U. LANGER, *A Tutorial on Elliptic PDE's and Their Parallelization*, Software, Environment, and Tools series, SIAM Books, Philadelphia, 2003.
- [7] C. C. DOUGLAS, J. HU, J. RAY, D. T. THORNE, AND R. S. TUMINARO, *Fast, adaptively refined computational elements in 3D*, in Computational Science – ICCS 2002, vol. 3, Berlin, 2002, Springer–Verlag, pp. 774–783.
- [8] S. A. DUDEK AND P. COLELLA, *Steady-state solution-adaptive euler computations on structured grids*, Tech. Rep. LBNL-41343, Lawrence Berkeley National Laboratory, 1998. Also presented at the 36th AIAA Aerospace Sciences Meeting and Exhibit, January 1998, Reno, NV, paper no. AIAA-98-0543.
- [9] D. MARTIN AND K. CARTWRIGHT, *Solving Poisson's equation using adaptive mesh refinement*. <http://seesar.lbl.gov/anag/staff/martin/tar/AMR.ps>, 1996.
- [10] S. F. MCCORMICK, *The fast adaptive composite (FAC) method for elliptic equations*, Math. Comp., 46 (1986), pp. 439–456.
- [11] K. W. MORTON AND D. F. MAYERS, *Numerical Solution of Partial Differential Equations.*, Cambridge University Press, Cambridge, 1994.
- [12] J. J. QUIRK AND S. KARNI, *On the dynamics of a shock-bubble interaction*, J. Fluid Mech., 318 (1996), pp. 129–163.
- [13] J. RAY, C. KENNEDY, S. LEFANTZI, AND H. N. NAJM, *High-order spatial discretizations and extended stability methods for reacting flows on structured adaptively refined meshes*, in Third Joint Meeting of the U.S. Sections of The Combustion Institute, Chicago, Illinois, March 2003.

- [14] U. RÜDE, *Mathematical and Computational Techniques for Multilevel Adaptive Methods*, vol. 13 of Frontiers in Applied Mathematics, SIAM, Philadelphia, 1993.
- [15] R. SAMTANEY, S. JARDIN, P. COLELLA, AND T. LIGOCKI, *High-resolution adaptive mesh simulations of magnetic reconnection*, in Bulletin of the American Physical Society, Division of Plasma Physics, American Physical Society, 2002. DPP meeting, Orlando, FL, Nov 11-15, 2002.
- [16] D. T. THORNE, *Multigrid with Cache Optimizations on Adaptive Mesh Refinement Hierarchies*, PhD thesis, University of Kentucky, Department of Computer Science, Lexington, KY, December 2003.
- [17] R. S. VARGA, *Matrix Iterative Analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1962.