CACHE AWARE MULTIGRID ON AMR HIERARCHIES*

DANNY THORNE†

Abstract. A cache optimized multilevel algorithm to solve variable coefficient elliptic boundary value problems on adaptively refined structured meshes is described here. The algorithm is optimized to exploit the cache memory subsystem. Numerical results are given demonstrating the efficiency of the cache optimization.

1. Introduction. This paper presents a combination of adaptive refinement [2, 3, 10] and multilevel [4, 6, 8] procedures to solve variable coefficient elliptic boundary value problems of the form

$$\begin{cases} \mathcal{L}(\phi) = \rho \text{ in } \Omega, \\ \mathcal{B}(\phi) = \gamma \text{ on } \partial\Omega, \end{cases} \tag{1.1}$$

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, notation is borrowed from the adaptive mesh refinement (AMR) community.

The focus of this research is on the effects of cache aware algorithms on multigrid in an AMR context. Cache aware algorithms are designed to minimize the number of times data goes through cache, thereby increasing the efficiency of the algorithm. The efficiency of the algorithm is improved because cache memory is much faster than main memory, so the CPU can be kept more busy when it is getting data from cache memories.

2. 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 Ω is overlaid by a union of tensor product meshes $\Lambda^{1,j}$, $j = 1, \ldots, n_1$ that form a grid in Ω:

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

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 defined throughout this paper. We define the domains Ω^ℓ and $\Omega^{\ell,j}$ as the minimum domains that include Λ^ℓ and $\Lambda^{\ell,j}$, respectively. Normally, Ω^ℓ is a union of disconnected subdomains (one subdomain corresponding to each level ℓ patch). Note that within an adaptive grid refinement code ℓ_{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

$${\Lambda_c}^\ell = \bigcup_{i=1}^{\ell} (\Lambda^i - \mathcal{P}(\Lambda^{i+1})), \tag{2.1}$$

where $\mathcal{P}$ is the projection operator discussed below. The ℓ^{th} composite grid, ${\Lambda_c}^\ell$, contains all points from the ℓ^{th} level patches, Λ^ℓ, 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.

*This work was supported in part by grants from Sandia National Laboratories, the National Science Foundation (grants DMS-9707040, ACR-9721388, and CCR-9988165) and gifts from the Intel and Hewlett-Packard Corporations.

†University of Kentucky, Department of Computer Science, 325 McVey Hall, Lexington, KY 40506-0045, USA, thorne@ccs.uky.edu.

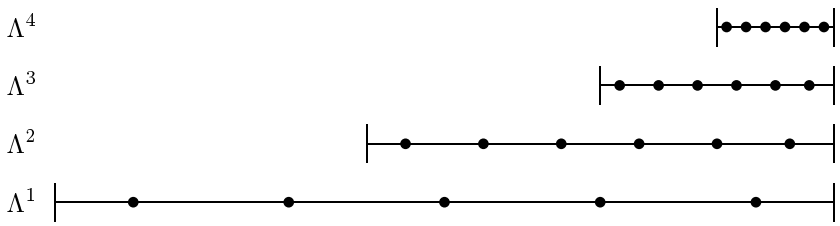

FIGURE 2.1. *A sample grid hierarchy.*

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 Ω^ℓ or grids Λ^ℓ. In terms of domain and grid superscripts, $\mathcal{P}$ *projects* from "fine to coarse," i.e., $\ell \to \ell - 1$ and $\mathcal{R}$ *refines* from "coarse to fine," i.e., $\ell \to \ell + 1$.

We require nesting of domains

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

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 §3.2. 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 derivate (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 §3.1 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

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

that coarse-fine interfaces never exist between more than one level of refinement. This ensures that the flux matching procedure is well defined and of the right approximation order near patch boundaries in the interior of Ω [6].

We approximate the solution to (1.1) using a multigrid algorithm to numerically solve the finite dimensional problem

$$\mathcal{L}_c^{\ell_{max}} \phi_c^{\ell_{max}} = \rho_c^{\ell_{max}}, \tag{2.2}$$

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, the implementation is designed to be patch based (see §4). A simple 1D AMR patch hierarchy is illustrated in Figure 2.1. 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.

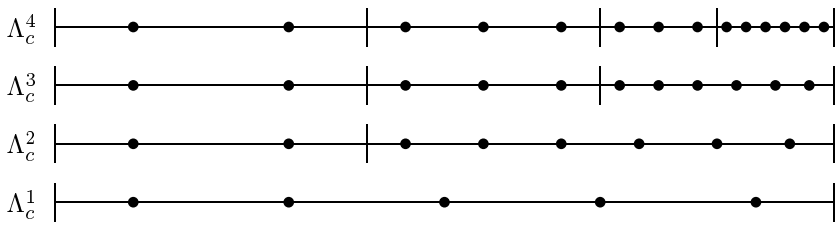

FIGURE 2.2. *Composite grid hierarchy corresponding to the sample grid in Figure 2.1.*

There can be multiple patches per level, although the grid hierarchy in Fig. 2.1 has only one patch per level. Figure 2.2 illustrates the corresponding composite grid hierarchy for the simple 1D example. Each level in this composite grid hierarchy is a composite grid defined in (2.1). Equation (2.2) 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 Λ_c^ℓ for all ℓ.

As mentioned above, the implementation is designed to be patch based, and so §4 employs patch based versions $\mathcal{L}^\ell$ and $\mathcal{L}^{nf,\ell}$ of the discrete operator in Alg. 1. In simple terms, the patch based versions of the discrete operator are merely restrictions of the composite grid operators. A more precise definition of the patch base operators is facilitated by the tools constructed in the next section.

3. Tools for AMRMG. 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 (1.1) that this discussion will address is

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

where Ω is a rectangular domain and γ is a constant. In addition, this example will assume isotropic mesh spacing $h = \Delta x = \Delta y$. The constant coefficient case, e.g. $a = 1$, is discretized using finite differences. The variable coefficient case, e.g. $a = a(x, y)$, is discretized using a control volume approach [9].

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.

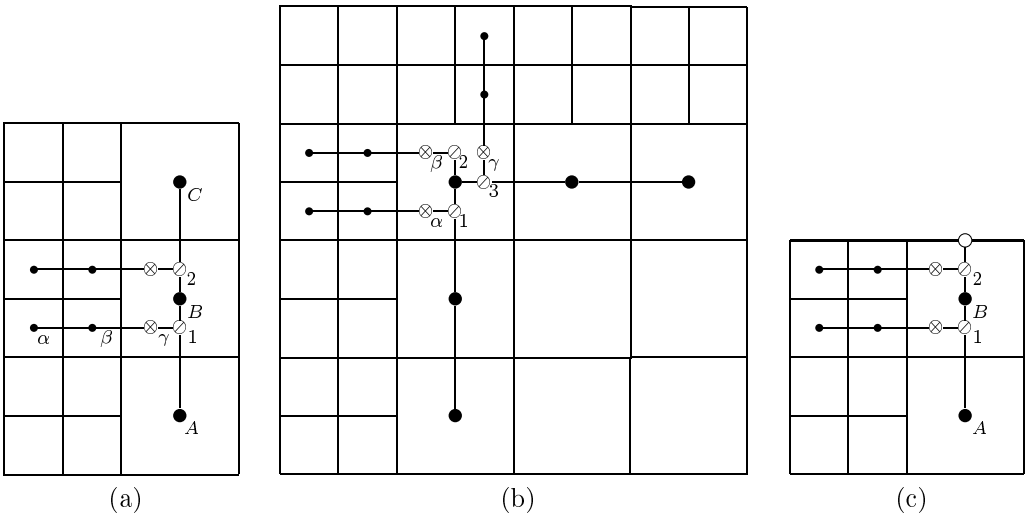

FIGURE 3.1. *Interpolation of Ghost Points. The dark edge in (c) indicates a boundary.*

3.1. Ghost Point Interpolation. 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.

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 3.1(a). There are two steps.

Step 1: Compute values for the the $\oslash$ 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

$$c_0 + c_1 x + c_2 x^2 = u(x),$$

where

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 2h & (2h)^2 \\ 1 & 4h & (4h)^2 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \Rightarrow \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} a \\ \frac{1}{h}(b - \frac{1}{4}c - \frac{3}{4}a) \\ \frac{1}{h^2}(\frac{1}{8}c - \frac{1}{4}b + \frac{1}{8}a) \end{bmatrix}.$$

Then

$$u(\oslash_1) = c_0 + c_1(\frac{3}{2}h) + c_2(\frac{3}{2}h)^2,$$

and

$$u(\oslash_2) = c_0 + c_1(\frac{5}{2}h) + c_2(\frac{5}{2}h)^2.$$

Step 2: Compute values for the $\otimes$ points. Let $a = u(\bullet_\alpha)$, $b = u(\bullet_\beta)$ and $c = u(\oslash_1)$, and let h be the mesh spacing on the finer grid. Again, we derive a C^1 quadratic interpolation formula using

$$c_0 + c_1 x + c_2 x^2 = u(x),$$

where

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & h & h^2 \\ 1 & \frac{5}{2}h & (\frac{5}{2}h)^2 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \Rightarrow \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} a \\ -\frac{1}{15h}(21a - 25b + 4c) \\ \frac{2}{15h^2}(3a - 5b + 2c) \end{bmatrix}.$$

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 3.1(b). Again, values are needed at the $\otimes$ points, and there are two steps. The notation $c_{\{0,1,2\}}$ 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 coefficents $c_{\{0,1,2\}}$ needed to compute values for the $\oslash$ 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 coefficents $c_{\{0,1,2\}}$ 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 3.1(c). Once again, values are needed at the $\otimes$ points, and there are two steps.

Step 1: Compute values for the $\oslash$ points as in *Step 1* of the previous sections. Use $a = u(\bullet_A)$, $b = u(\bullet_B)$ and $c = u(\bigcirc)$.

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

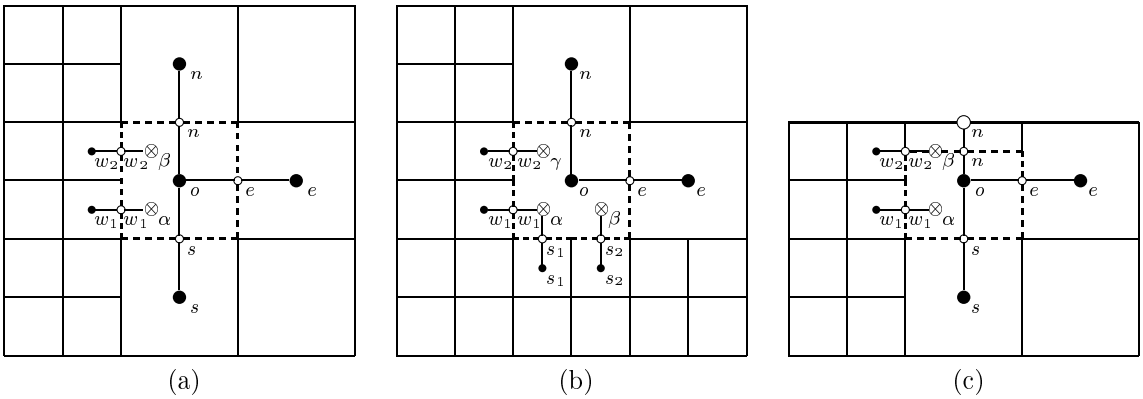

FIGURE 3.2. *Flux matching for the variable coefficient case. The dark edge in (c) represents a boundary.*

3.2. Flux Matching. 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.

See Fig. 3.2 for illustrations. The regions within the dashed lines are the control volumes, denoted by V. The boundary of the control volumes, represented by the dashed lines, are denoted by ∂V.

The grid points on edges represent coefficient values. The grid points in cells represent solution values, as usual. For example, $a_n = a(\circ_n)$ and $u_n = u(\bullet_n)$. An exception is the grid point in the control volume which represents a solution value *and* a coefficient value: $a_o = a(\bullet_o)$ and $u_o = u(\bullet_o)$.

Discretizing the integral form

$$-\int_{\partial V} a\frac{\partial u}{\partial n} = \int_V \rho$$

gives

$$-(f_n - f_s + f_e - f_w) = h^2\rho_o,$$

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

3.2.1. One Coarse-Fine Interface. See Fig. 3.2(a). The fluxes for this case are

$$\begin{aligned} f_n &= a_n(u_n - u_o), \quad & f_s &= a_s(u_o - u_s), \\ f_e &= a_e(u_e - u_o), \quad & f_w &= a_{w_1}(u_\alpha - u_{w_1}) + a_{w_2}(u_\beta - u_{w_2}). \end{aligned}$$

3.2.2. Two Coarse-Fine Interface. See Fig. 3.2(b). The fluxes here differ from §3.2.1 only in the south flux:

$$f_s = a_{s_1}(u_\alpha - u_{s_1}) + a_{s_2}(u_\beta - u_{s_2}).$$

3.2.3. At a Boundary. See Fig. 3.2(c). The fluxes for this case are

$$\begin{aligned} f_n &= 2a_n(u_n - u_o), \quad & f_s &= a_s(u_o - u_s), \\ f_e &= \tfrac{3}{4}a_e(u_e - u_o), \quad & f_w &= a_{w_1}(u_\alpha - u_{w_1}) + \tfrac{1}{2}a_{w_2}(u_\beta - u_{w_2}). \end{aligned}$$

4. AMRMG. 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 ℓ with the interfaces between coarse and find 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 Λ^ℓ, ignoring finer levels and, hence, not requiring the flux matching procedure.

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 Λ^ℓ
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 Λ^ℓ
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 Λ^ℓ
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 Λ^ℓ
18: $\bar{e}^\ell \leftarrow 0$
19: $\bar{e}^\ell \leftarrow S^\ell(\bar{e}^\ell, r^\ell)$ on Λ^ℓ
20: $e^\ell \leftarrow e^\ell + \bar{e}^\ell$ on Λ^ℓ
21: $\phi^{\ell_{max}} \leftarrow \phi^{\ell_{max}} + e^\ell$ on $\Lambda^\ell - \mathcal{P}(\Lambda^{\ell+1})$

4.1. Post-smoothing only. Alg. 1 can be sped up considerably by only doing post-smoothing. There are several advantages to this approach:

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.

Doing post-smoothing only is a substantial change over [6] and Alg. 1. It is especially useful when implementing cache aware algorithms as discussed in §5. One can expect better cache effects when more smoothing iterations are done consecutively. Plus, 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. Furthermore, in the post-smoothing only version, the residual converges faster due to the fact that pre-smoothing has an exacerbating effect on the residual at coarse-fine interfaces in the initial iterations [7].

5. Cache Optimizations. 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 ℓ rows, and let m be the number of smoothing iterations required. It is necessary for $\ell + m - 1$ 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 ℓ rows of the grid. The data associated with rows 1 to ℓ 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.

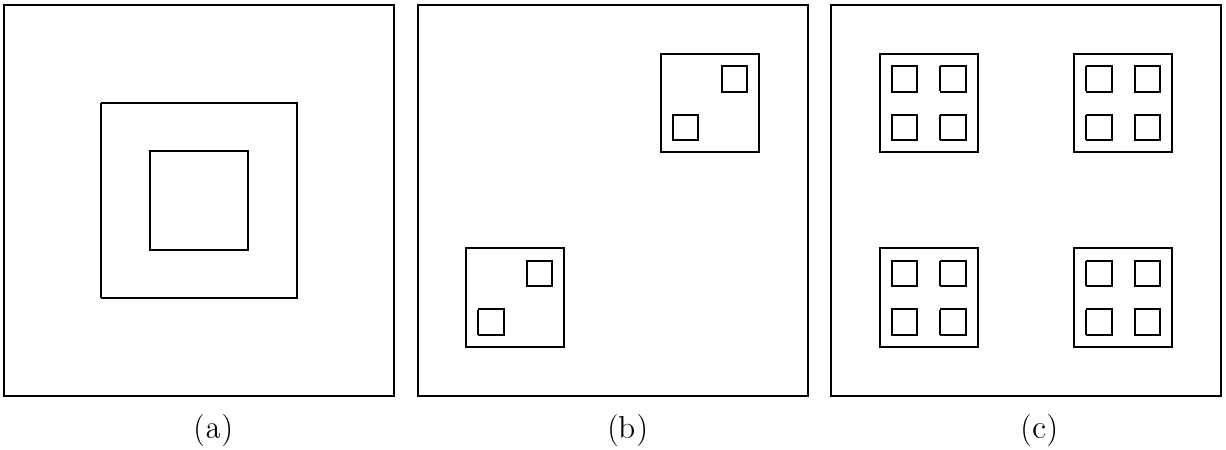

FIGURE 6.1. *(a) One refinement per patch. (b) Two refinements per patch. (c) Four refinements per patch.*

The second case is for the rest of the blocks of ℓ 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^{th} update in the second block, we can go back and update rows ℓ 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.

5.1. Multigrid with the residual computation integrated into the smoother. 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 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.

5.2. Effects of the Coefficient Matrix. The coefficient matrix has an effect on the cache optimizations since it has to go through cache along with the solution and right hand side. In an attempt to minimize the effect, we tried storing the right hand side in the coefficient matrix in an interleaved manner. Hence, for a given grid point, the coefficients and the right hand side needed to update that point are contiguous in memory. However, the split residual computation complicates this approach. On every level, there are parts of the domain where the residual has to be computed from the right hand side *as well as* parts of the domain where the residual is projected from finer levels. Depending on the context, the right hand side entries in the coefficient matrix might need to be either the original right hand side or the current residual. Updating the state for a given context involves copies which slow the algorithm. The results in the AMR context are much better when the right hand side (and residual) are left in separate memory from the coefficient matrix.

6. Numerical Results. This section shows results on the hierarchies illustrated in Fig. 6.1 and also on a full domain refinement hierarchy. The refinement patters are not meaningful to the nature of the problem being solved here. The are contrived merely to demonstrate the behavior of the algorithm. The AMR base grid in 6.1(a) is 128×512 grid points, and there are three levels of refinement above that (although only two are illustrated). The AMR base grid in 6.1(b) and 6.1(c) is 256×1024 grid points, and there are four levels of refinement above that (although only two are illustrated). Geometric multigrid is used as the solver on the AMR base grid. The full domain refinement case starts on an 8×32 base grid and has a total of 7 grid levels.

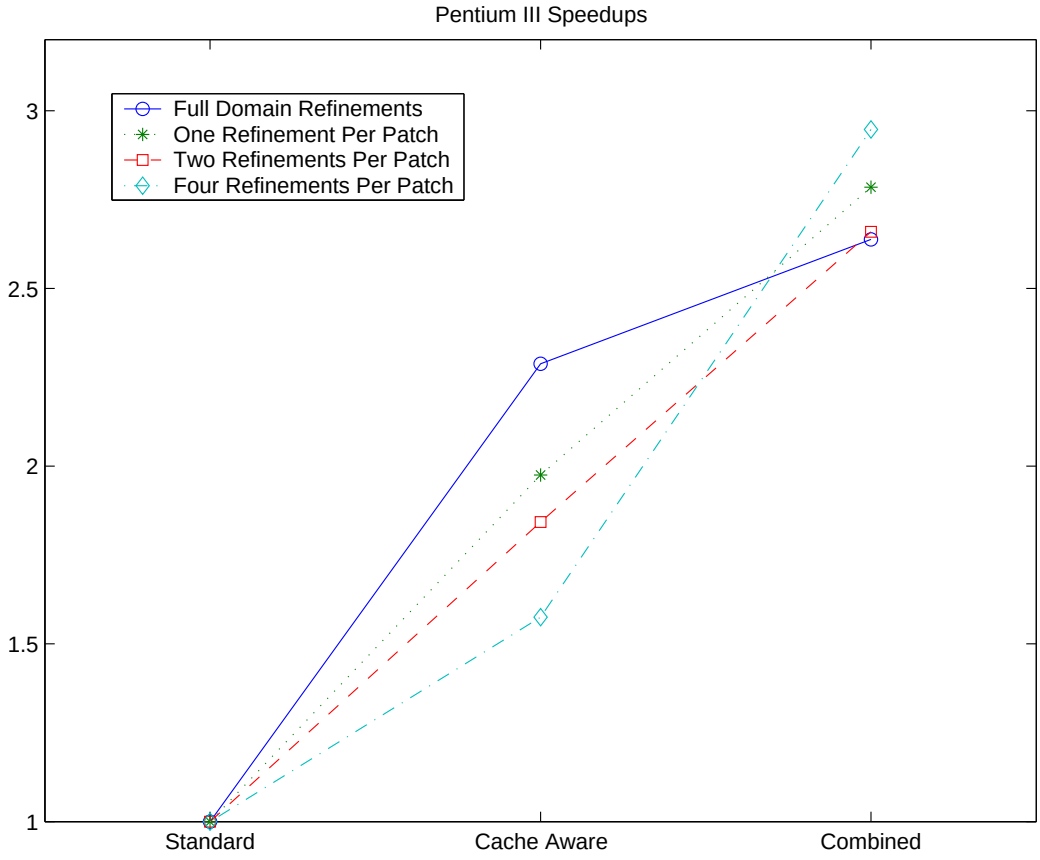

FIGURE 6.2. *Pentium speedups.*

The graphs show results for the AMR multilevel method emplying a *standard* implementation of the smoother, the *cache-aware* smoother, and the *combined smoother* (as described in §5.1).

The base grid discretizes the rectangle $[0,1] \times [0,4]$. The right hand side of the Poisson equation is chosen so that the solution is $u(x,y) = \sin(\pi x)\sin(\pi y/4)xe^{x^2+(y/4)^2}$ and the coefficient is $a(x,y) = 1 + \sin(\pi x)\sin(\pi y/4)xe^{x^2+(y/4)^2}$. The initial guess on the base grid is $u = 0$, and the convergence criteria is $||r_c|| < 10^{-6}||\rho_c||$.

Fig. 6.2 show the results of the set of experiments run on a Pentium III. Fig. 6.3 show the results of the set of experiments run on an Itanium 1. Fig. 6.4 show the results of the set of experiments run on a Itanium 2. These experiments show the speedups associated with the cache optimizations. The timings measure only the solution procedure. They do not include initialization of the hierarchy, coefficient matrix and right hand side. The total speedups are between 2 and 3 times on all three machines, except for one case on the Itanium 1 which exceeds 3 by a little.

The speedups (not shown) for doing post-smoothing only versus doing pre-smoothing *and* post-smoothing are around 20% with the cache aware smoother. Note that both cases do the same total number of smoothing iterations per level, 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 of the V. The post-smoothing only case uses 4 smoothing iterations on only one side of the V.

6.1. Conclusions. We experimented with cache aware smoothers and integration of the residual computation with the smoother. Both of these give good speedups. The additional speedups for the integrated residual computation are particularly remarkable for the AMR hierarchies.

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.

Future plans for this research include experimenting with three dimensional problems and parallel algorithms. Progress is already being made toward both of those topics. In addition, we are going to experiment with a variety

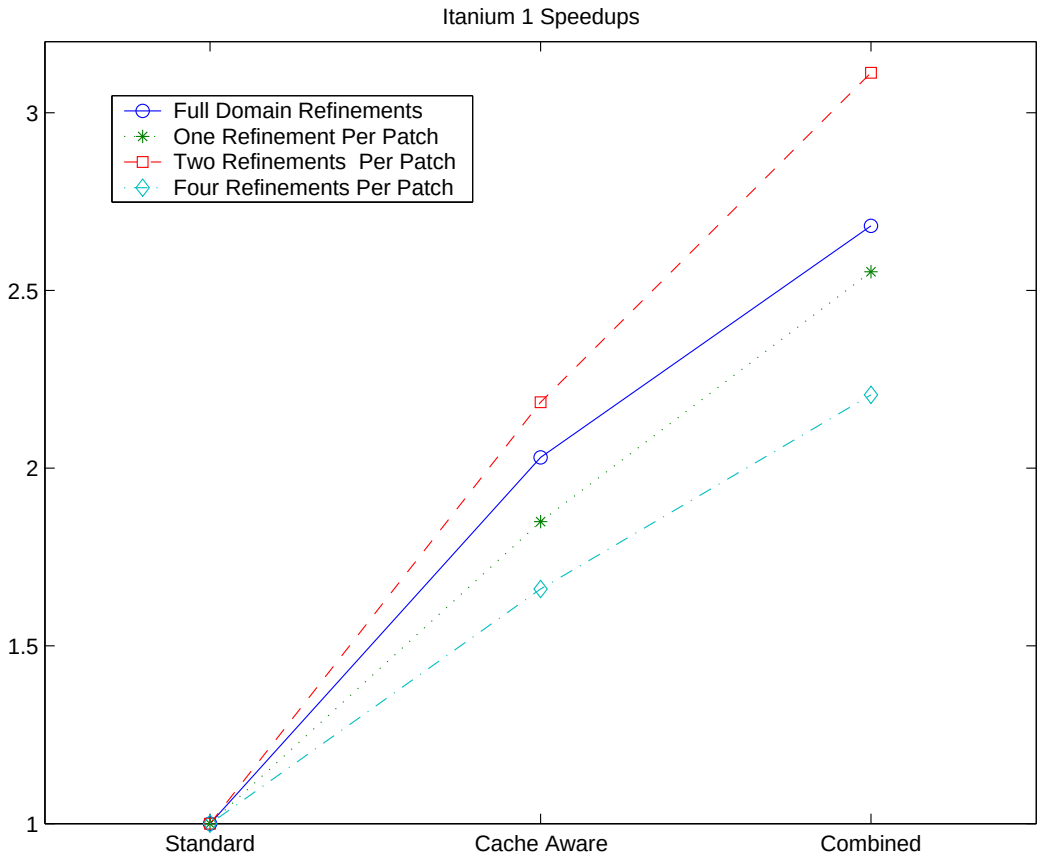

FIGURE 6.3. *Itanium 1 speedups.*

of other cache optimization techniques (beyond mere row based blocking). The 3D case, in particular, needs more sophisticated approaches, such as those discussed in [5].

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.*, 82:64–84, 1989.

[3] M. J. Berger and J. Oliger. An adaptive mesh refinement for hyperbolic partial differential equations. *J. Comput. Phys.*, 53:484–512, 1984.

[4] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A Multigrid Tutorial*. SIAM Books, Philadelphia, 2000. Second edition.

[5] 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,*, volume 3, pages 774–783, Berlin, 2000. Springer–Verlag. Check publisher.

[6] D. Martin and K. Cartwright. Solving Poisson's equation using adaptive mesh refinement. http://seesar.lbl.gov/anag/staff/martin/tar/AMR.ps, 1996.

[7] D. F. Martin. *An Adaptive Cell-Centered Projection Method for the Incompressible Euler Equations*. PhD thesis, University of California at Berkley, 1998.

[8] S. F. McCormick. The fast adaptive composite (FAC) method for elliptic equations. *Math. Comp.*, 46:439–456, 1986.

[9] K. W. Morton and D. F. Mayers. *Numerical Solution of Partial Differential Equations*. Cambridge University Press, 1994.

[10] U. Rüde. *Mathematical and Computational Techniques for Multilevel Adaptive Methods*, volume 13 of *Frontiers in Applied Mathematics*. SIAM, Philadelphia, 1993.

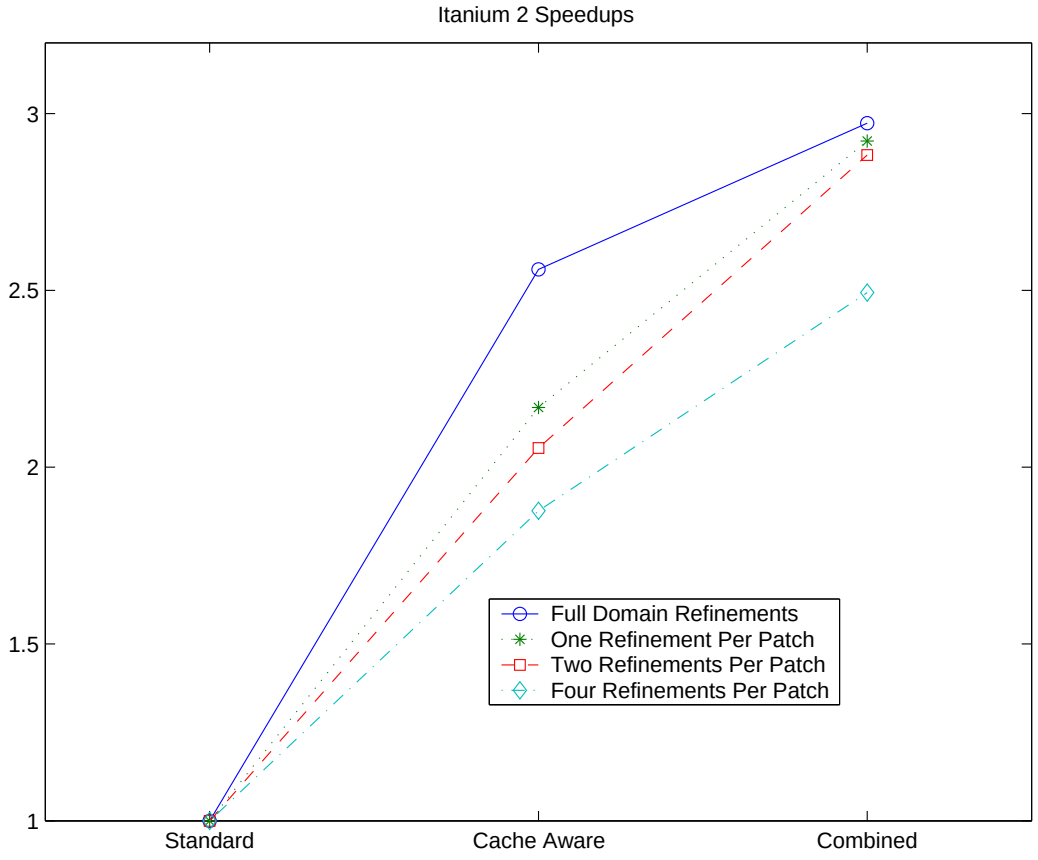

FIGURE 6.4. *Itanium 2 speedups.*