Multigrid Methods in Science and Engineering

Craig C. Douglas
IBM T.J. Watson Research Center and Yale University

By combining computation from several scales of mesh fineness, multigrid and multilevel methods can improve speed and accuracy in a wide variety of science and engineering applications. This tutorial sketches the history of the techniques, explains the basics, and gives pointers to the literature and current research.

The concept of multigrid and multilevel methods is not terribly new. In fact, one variant predates electronic computers by decades. Starting in the late 1970s, however, theoretical and practical developments converged to make these methods important and widespread computational techniques. Multigrid and multilevel methods are now used in aerospace simulation (flow over an airplane, missile, or space shuttle), petroleum engineering (reservoir or pipeline simulation), environmental studies (tracking pollution and hurricanes, ocean modeling, weather prediction), combustion, computers (video signal display), and other fields.

But what are they?

When, as frequently happens in practical science and engineering problems, a function is too complicated to evaluate with analytical calculus, we can approximate a solution on a grid by solving discrete, finite-dimensional problems.

First-year calculus books show the simplest example of this while explaining analytical calculus itself: fitting rectangles under a curve to approximate an integral. The rectangles are a type of grid. The narrower they are the more accurate the solution. As their width approaches zero the function becomes continuous and the result exact. But grids with discrete elements of finite size introduce errors. The larger the elements—the coarser the grid—the larger the errors usually are.

Multigrid takes its name from using multiple grids and finite-dimensional problems to approximate the solution to the original problem. (The terms multi*grid* and multi*level* are closely related and often identical; though multilevel techniques can also be used on problems that do not involve a grid per se.)

Multiple grids

Some computing strategies make the grid or mesh very fine, either all over or in some critical places, to get better accuracy, but this can take tremendous computing power. There is another way—using *multiple* grids of differing fineness and combining the results in the right way.

The fundamental idea behind all multigrid methods is to combine computations done on different scales, using results from one scale to obliterate the error components characteristic of another. Error components of a numerical computation can be thought of (in Fourier space) as waves made up of linear combinations of scaled sine and cosine functions.

1070-9924/96/$5.00 © 1996 IEEE

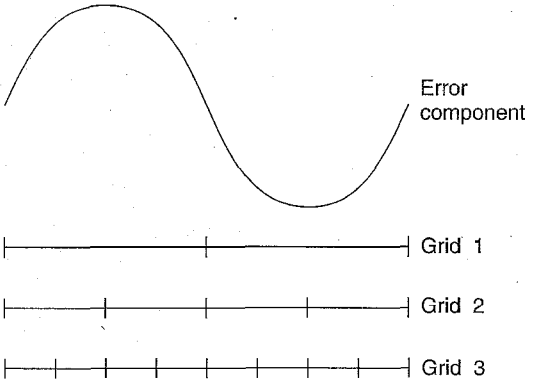

Figure 1. Wavelength scales on different grids.

A long wave, which has low frequency on a fine grid, becomes a shorter wave with higher frequency when represented on the scale of a coarser grid (see Figure 1). What takes four grid points to represent on level 3 takes only two grid points on level 2, and only one grid point on level 1. A number of iterative methods (such as relaxation and alternating-direction implicit methods) are known to be very efficient at damping short-wave error components, but less so at damping long-wave components. Given this fact, the coarse grid with its short waves becomes more useful.

Finer grids provide a more accurate approximation to the original mathematical problem than coarser grids. However, if the coarser grid is *fine enough*, information about the solution can be interpolated from it onto the finer grid to accelerate the solution process there. Thus we have what might seem counterintuitive to some: we actually use a coarse-grid solution to correct a fine-grid solution.

Multigrid originally provided a cheap, accurate initial guess to use in solving the problem on a finer grid. It became a fancy form of iterative improvement in the 1960s.

For those wanting to probe further, Briggs gives a particularly well written motivation for multigrid methods.[1] More detailed books include those by Hackbusch,[2] Wesseling,[3] and Bramble.[4] A related area is domain decomposition, which is reviewed in detail by Smith et al.[5]

Multigrid methods are difficult to use on nontrivial problems. Yet they flourished during the 1980s and 1990s and became quite standard. This is due to their rapid convergence rates and their efficiency (as measured by their operation count in the familiar big-O notation). People came to realize that if a "unilevel" code was well designed, adding a multilevel procedure to it was not as difficult as it might appear, and worth the effort.

While multigrid is not the first procedure for solving elliptic partial differential equations in optimal time, it is the first do so stably and to also use an optimal order of memory space. This has led the multigrid community to extend these methods to many problems besides elliptic PDEs.

When solving very complicated problems, though, it should be noted that multigrid does not usually run in the linear runtime commonly associated with, say, Poisson's equation on a square with a uniform mesh. For many problems, multigrid lowers the operation count to $O(N^i \log N)$ from $O(N^{i+j})$ or $O(N^{i+j} \log N)$, for $j \geq 1$, where N is the number of unknowns and i and j are small numbers. However, for problems with hundreds of thousands or millions of unknowns, this is still quite a speedup.

Historical perspective

Multigrid or multilevel methods have an interesting history. At several points, a new collection of people entered the field simultaneously and made new discoveries.

Before 1960: Personnel computing

In the era of "personnel computing" (mostly before 1960), a common methodology was to collect a group of people into a room and assign part of a domain to each person. People computed using relaxation methods. A common procedure for generating the initial guess was to solve the problem first on a much coarser grid with about one tenth as many grid points. This coarse-grid approximate solution was interpolated onto the fine grid. Then the personnel computers began a "parallel computation" to solve the problem on the fine grid. In his 1940 book, Southwell describes such a process as common in British aeronautics companies in the 1920s.[6]

Hence, parallel multigrid is a natural progression from single-processor multigrid. After all, it is just doing what people did for years by hand.

1961–1971: The Soviet school

The Soviet school of numerical mathematics produced four major papers on multigrid during this period. The West essentially ignored these papers for many years until Achi Brandt popularized the method during the mid-1970s.

The 1961 and 1964 papers by Fedorenko described in complete detail a multigrid procedure for solving Poisson's equation on a square with uniform grid elements.[7,8] The procedure has a resemblance to iterative improvement for Gaussian elimination. A central difference dis-

cretization was employed. The convergence rate was determined using Fourier analysis that included the boundary conditions.

The 1966 paper by Bakhvalov analyzed general, second-order, variable coefficient elliptic problems on a square domain with a uniform mesh.[9] A correction scheme (see later) was analyzed analytically. For N unknowns, this method was proven to converge to the order of the truncation error in work proportional to $N \log N$. Had Bakhvalov started from the coarsest level instead of the finest one, he would probably have discovered the optimal-order work estimate later associated with Brandt.

The 1971 paper by Astrakhantsev analyzed a finite-element multigrid method.[10] Once again, the optimal-order work estimate was just missed. There is quite a similarity between this paper and a later one by Bank and Dupont.[11]

Back in the West a similar set of algorithms, known as aggregation-disaggregation methods, were becoming common tools in economics. These methods eventually were applied to problems in nuclear reactors, circuit simulation, and electrical power networks.

Middle to late 1970s

Achi Brandt is known as the father of multigrid methods, with good reason. He was one of the first to recognize their potential and was willing to stand on desk tops (literally) at conferences and point out their advantages in a way that got people's attention. Many still consider Brandt's 1977 paper[12] the origin of modern multigrid.

One of Brandt's contributions is the application of local-mode analysis to estimating the convergence rate of a multigrid algorithm. By convergence rate, we mean the rate at which the error is reduced from one iteration to the next. Local-mode analysis is not rigorous, but frequently gives a good enough estimate.

The work of several theoreticians during this era still stands out. Hackbusch, Wesseling, Hemker, Bank, and Nicolaides all wrote or cowrote pivotal papers. Many of these were written at about the same time with no knowledge of the others' works. (Unlike today, preprints were quite difficult to acquire then.) Nonetheless, the results were in some cases quite similar.

The big 80s

Multigrid's golden age was the 1980s. A tremendous amount of theory was published, and multigrid became a standard solution method in many application areas. Multigrid practitioners gelled into a fairly cohesive community.

Four sets of multigrid conferences began during the 1980s: the European multigrid conferences, the Copper Mountain conferences, the GAMM workshops (in East Germany for many years), and the Oberwolfach workshops. Each provided a wealth of papers and advances that continue to this day. These meetings and their related publications were the single largest advance in the field in this era.

The field was extended into areas like hyperbolic and parabolic differential equations. In particular, several theoretical tools became common during this period.

In my dissertation (later published elsewhere[13]) I provided a tool for analyzing the convergence rate abstractly without having to use a different proof for each of several discretizations. This led to a paper by Bank and me which produced a theoretical tool for sharply estimating convergence rates for nontrivial problems.[14]

Papers by Braess, Mandel, McCormick, Parter, Verfürth, and Yserantant extended the known theory to numerous multigrid algorithms and model problems without regard to the machine architecture. A number of parallel multigrid algorithms were developed, analyzed, and tested on actual machines during this period.

Hierarchical-basis multigrid[15] became a useful tool, particularly in two-dimensional finite-element multigrid solvers based on adaptively refined meshes. In many cases this is a slower solution method than more standard adaptive-mesh multigrid methods, but it has the distinct advantage of being provably convergent. Algebraic multilevel iterative methods (AMLI), first really

More Information: MGNet and the Multigrid Digest

Preprints, conference proceedings, computer packages, and a large BibTex database of references are maintained on MGNet, the multigrid community's Internet special interest group.

To subscribe to the Multigrid Digest, a monthly MGNet electronic newsletter, send an e-mail message containing your preferred e-mail address to mgnet-requests@cs.yale.edu.

MGNet, located at Yale University and mirrored at CERFACS, can be accessed through the World Wide Web at

- http://casper.cs.yale.edu/mgnet/www/mgnet.html
- http://www.cerfacs.fr/~douglas/mgnet.html

An annotated guide to many other multigrid WWW sites appears on MGNet.

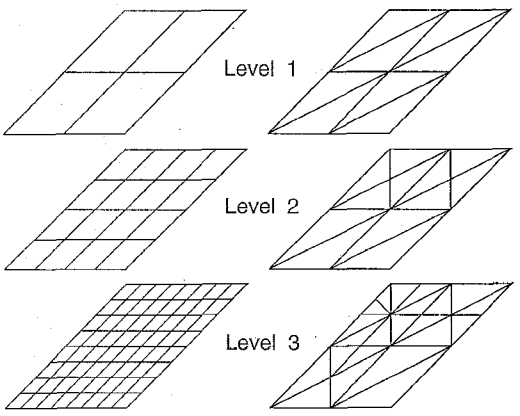

Figure 2. Examples of grids on three levels.

developed at the University of Colorado at Denver and the GMD in St. Augustin in the 1980s (see the book chapter by Ruge and Stüben[16] and the references therein), has become a very interesting theoretical and possibly computational area in the 1990s. The 1989 paper of Axelsson and Vasselevski[17] started a slow but continuing progression of ideas and papers (resulting in an interesting 1996 conference on the topic[18]).

The 1990s: High performance needed

A significant portion of multigrid research in the 1990s so far has been motivated by three computational developments that have profoundly changed applied mathematics in general.

The first was the easy availability of extremely fast, relatively inexpensive RISC-based workstations. Suddenly, people who did not have $30 million computer budgets could do very high performance computing at their office or home.

The second was the appearance of reliable parallel computers.

Suddenly, many more people were doing truly high-performance computing. Many of them were aware of multigrid methods, but had had no reason to use them yet. Soon, though, the new large problems outdistanced the new machines. When a problem takes a CPU day or week by conventional methods, multigrid becomes interesting.

The third development was the explosion of information on the Internet, including the World Wide Web. In 1991, MGNet began as a monthly digest for multigrid and domain decomposition issues, and quickly became a central resource for storing codes, preprints, conference proceedings and announcements, and literature citations.[19] (See "More Information" sidebar.)

While the computational world changed, some very interesting work was done in both the multigrid and domain decomposition communities trying to determine when it makes sense to use multigrid inside of a domain decomposition procedure or vice versa (more on this later).

Theory progressed in the areas of nonconforming finite-element methods, methods and analysis for nonlinear elliptic problems, multilevel domain decomposition methods, and algebraic multilevel methods. Multigrid methods became quite common in extremely complicated application areas.

In another 1990s development, Bramble, Pasciak, Wang, and Xu developed a *regularity-free* theory.[20] This usually requires a specific symmetry in the solvers with respect to an inner product. Since this eliminates the standard variant of Gauss–Seidel, many did not see this theoretical method as an advantage. However, regularity-free theory is highly useful for proving the convergence of multigrid.

What do multilevel algorithms look like?

Before giving mathematical definitions, let's see how multilevel algorithms actually take advantage of computation on different scales.

Different scales

What is meant by computing on different scales? Some simple examples arise when the grids have different mesh spacings. The left side of Figure 2 shows a uniformly refined set of grids. The scales are clearly differentiated by a factor of two. Hence, what takes four grid lines to represent on level 3 takes only two on level 2 and one on level 1.

On the right in Figure 2 is a set of adaptively (nonuniformly) refined grids. This type of grid refinement usually is seen only in finite-element multigrid solvers. The scales are different only where the solution is not adequately resolved on a coarse grid.

For adaptively chosen grids, there are several variants of multigrid. The standard approach used to be to have basis functions for all the elements on a given level. By carefully monitoring how many new unknowns are added when creating a new, refined grid, this method is computationally useful. Otherwise, usually not enough new unknowns are added and the multigrid solver spends most of its time changing levels instead of approximately solving problems.

Another variant is to have basis functions for only the new elements, thus forming a hierar-

chical set of basis functions. This technique works well in one and two dimensions but not in three or more, since the matrix bandwidth and number of nonzeroes become excessive.

W.F. Mitchell's dissertation provides a hybrid algorithm for adaptive-grid multigrid which captures the best features of the first two variants.[21]

A third approach is to use domain decomposition and to have locally uniform meshes which may overlap. This has many advantages. One is that most of these codes run fast on RISC-based workstations and vectorize well.

Another set of methods is being developed where only the finest grid (a completely unstructured one) is given, and the coarser grids must be constructed from it. Just removing vertices doesn't work since many elements then lose their triangular or tetrahedral shapes. A Delaunay triangulation procedure is frequently applied. This identifies which vertices will be in the new grid and produces triangles or tetrahedra to fit.

Implementing multigrid on unstructured grids leads to some serious data-structure issues. Unless great care is taken these codes run at the speed of the integer unit of a CPU, since floating-point operations comprise less than 10 percent of the total. Rüde covers this topic well.[22]

Among the multigrid community's disjoint groups, two that constantly conflict are those who refine given coarse grids and those who coarsen given fine grids. These groups tend to number their levels in the opposite order, confusing some newcomers. In this article I stick with the refining group's numbering scheme: finer levels have higher numbers, as in Figure 2.

Wandering between grid levels

One reason multigrid methods can be difficult to master is the field's terminology. Most of the jargon really describes quite common algorithmic and mathematical processes. From here on I'll assume that the reader is familiar with Detour 1, "So You Want to Speak Fluent Multigrid...," and the glossary. If a term confuses you, please look there.

If we are going to compute on different grid levels and combine the results, we need careful, well-reasoned procedures for how to do this. Let's start to look at the steps.

Multigrid algorithms come in two basic flavors. *Correction methods* start at the finest level, which we will call k, and use the coarser levels $j < k$ solely to compute a correction which is added to the approximate solution on level k. *Nested iteration methods* generate initial guesses on coarser levels and frequently reuse the coarser levels for corrections, too.

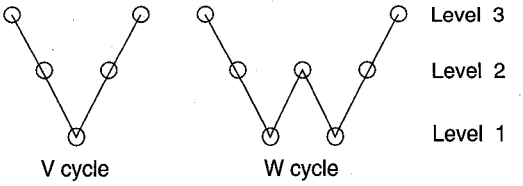

Figure 3. Multigrid algorithmic flow: correction methods. Level 3 has the finest grid, level 1 the coarsest. Computation flows from left to right in each type of cycle. (In these diagrams the finest level is traditionally shown at the *top*, unlike Figure 2.)

When deciding which flavor of multigrid algorithm to use for a real application, the following rule of thumb is key. *When a good initial guess is available, correction algorithms work well. Otherwise use an algorithm that generates an initial guess.*

The correction path. Two standard correction multilevel algorithms which arise in many different areas of multigrid are the *V cycle* and the *W cycle*. How computation moves from one level to another is pictured in Figure 3. On each level we use a different scale.

The V cycle starts on the finest level and traverses all the grids, one at a time, until it reaches the coarsest. Then it traverses them all again until it reaches the finest. From a recursive-algorithm viewpoint, the V cycle uses the coarser levels once per level for corrections. Later on, Figure 5 (and in even more detail, Table A in Detour 1) will explain this more.

The W cycle is similar except that it does two corrections per level. For many problems, a W cycle is considered more robust and easier to analyze theoretically than a V cycle. However, W cycles are much harder to parallelize efficiently because they spend more time on the coarser levels.

The nested iteration path. The other class of algorithms start on the coarsest level and wend their way to the finest. These are referred to as *nested iteration* methods since they contain standard correction algorithms (such as a V or W cycle) within them. Figure 4 shows how they move from level to level.

The oldest multilevel algorithm, a sort of degenerate nested iteration method, is the *one-way multigrid* algorithm (sometimes referred to as cascadic multigrid). As shown in Figure 4, it never computes a correction problem on a coarser level. The coarser levels' problems are solved sequentially and used (after interpolation) as the initial guess for the next finer level's prob-

Detour 1: So You Want to Speak Fluent Multigrid...

Multigrid has developed a language of its own (see also the glossary).

Terminology details

Suppose we're solving a linear PDE on a spatial domain Ω; perhaps a square. Say we're given a set of discrete grids or meshes Ω_j, $1 \le j \le k$, on Ω (for instance, see Figure 2). Based on the figure, the grids are indexed so that the fewest number of points is in Ω_1 (the *coarsest* grid) and there are more points as j increases. Hence, Ω_k has the most points and is the *finest* grid.

Hackbusch and Brandt introduced differing notations for the mapping operations of *restriction* and *prolongation* (see glossary), in which data are transferred from a grid Ω to the next coarser or finer grid. For the restriction mapping $\Omega_j \to \Omega_{j-1}$ Hackbusch used the symbol R_j and Brandt used I_h^{2h}. For the prolongation mapping $\Omega_{j-1} \to \Omega_j$ they used P_{j-1} and I_{2h}^{h}. Brandt's notation of $I_{\text{from}}^{\text{to}}$ clearly denotes the relationship between the mesh scales of neighboring grids in the mapping. (*From* and *to* are the mesh spacings of the source and target grids, respectively.) Obviously any constant could replace the 2.

On each grid, we can discretize the PDE to get a linear or nonlinear problem $A_j x_j = f_j$. Here A_j is the linear or nonlinear discrete operator, x_j is the approximate solution, and f_j is the right-hand side. There is at least one solver $S_j^{(i)}$ associated with the problem (where $i \ge 0$; normally 1 or 2). This is either a direct solver when $j = 1$ or an iterative method (for example, Gauss–Seidel or conjugate gradients) when $j \ge 1$.

Summarizing, the problem at level j has associated with it the components A_j, x_j, f_j, $S_j^{(i)}$, Ω_j, P_j, and R_j.

How does the multigrid process work?

Consider a simple V cycle. Figure 4 shows only a bare-bones schematic. The cycle is defined further, but somewhat abstractly, in Algorithm MGC. To see what is really going on, look at the details of a three-level V cycle as shown in Table A. The pre- and post-smoothers could be something as simple as Gauss–Seidel or as complicated as another multigrid solver on a collection of subproblems. One common option is for the pre-smoother to be the identity operator and the post-smoother to be a real iterative method (or vice versa). The smoother on level 1 could be either an iterative method or a direct solver such as sparse Gaussian elimination.

Table A. Steps in a multigrid three-level V cycle.

Step	Level(s)	Operation	Actually computes
0	3	Initialize	[matrix A_3, initial guess x_3^0, rhs f_3]
1	3	Solve	$x_3^1 \leftarrow$ pre-smooth (A_3, x_3^0, f_3)
2	3	Residual	$r_3 \leftarrow f_3 - A_3 x_3^1$
3	3→2	Project	$f_2 \leftarrow R_3 r_3$
4	2	Solve	$x_2^1 \leftarrow$ pre-smooth $(A_2, 0, f_2)$
5	2	Residual	$r_2 \leftarrow f_2 - A_2 x_2^1$
6	2→1	Project	$f_1 \leftarrow R_2 r_2$
7	1	Solve	$x_1^1 \leftarrow$ smooth $(A_1, 0, f_1)$
8	1→2	Interpolate	$u_2 \leftarrow P_1 x_1^1$
9	2	Solve	$x_2^2 \leftarrow$ post-smooth $(A_2, x_2^1 + u_2, f_2)$
10	2→3	Interpolate	$u_3 \leftarrow P_2 x_2^2$
11	3	Solve	$x_3^2 \leftarrow$ post-smooth $(A_3, x_3^1 + u_3, f_3)$

lem. While the number of operations is not of optimal order, one-way multigrid is surprisingly effective for hard engineering problems. (In fact, it has been the standard algorithm for combustion problems for at least 25 years.) One-way multigrid is the generalization of the method described by Southwell in 1940.[6]

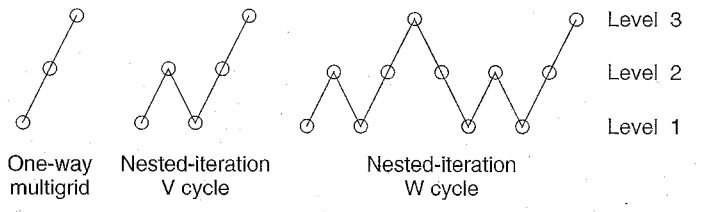

Figure 4. Multigrid algorithmic flow: nested iteration methods. Level 3 has the finest grid, level 1 the coarsest; computation flows left to right.

The nested iteration V cycles and W cycles are also quite common.

A charming aspect of the multigrid community is that each school may define these algorithms a little differently. For example, some would argue that the W cycle picture in Figure 3 should have two sets of corrections from level 3. You can also find disagreements over the form

(and the name) of the nested iteration W cycle pictured in Figure 4.

Multigrid algorithms

Now let's look at actual multigrid algorithms. Note that the V and W cycles and the nested iteration form the basis for defining multigrid algorithms to solve real problems. We can divide the algorithms into five general categories:

- algorithms for linear problems
- algorithms for nonlinear problems
- algorithms for time-dependent problems
- telescoping parallel algorithms
- nontelescoping parallel algorithms

It turns out that within each of these five categories there are two primary algorithms. Inside each primary algorithm may be a number of well-known variants. Heated discussions can arise over which primary algorithm is best for certain problems.

Linear problems

Fedorenko recognized multigrid first as a highly efficient algorithm for the numerical solution of simple elliptic partial differential equations, such as Poisson's equation on a square domain.

We can formulate linear problems into a sequence of linear systems of equations of the form

$$A_i x_i = f_i, \quad 1 \le i \le k. \tag{1}$$

For PDEs, the matrices A_i are derived using some discretization method (for example, finite differences, finite elements, finite volumes, or wavelet bases).

Suppose the order of each matrix A_i is N_i and that $N_i \le N_{i+1}$. In particular, many multilevel methods assume that

$$N_{i+1} \approx \sigma N_i, \quad \sigma \in \mathbb{R}, \quad \sigma \ge 1. \tag{2}$$

A common form for σ in d-dimensional problems is $C2^d$, where $C \in \mathbb{R}$ is determined by the discretization method and how the grid is refined ($C = 1$ is the most common, however).

Correction algorithms. The primary linear algorithm, a *correction algorithm* called MGC (see Figure 3 for some examples), is defined loosely in Figure 5. Again, look at Detour 1 if the terminology is new to you.

Algorithm MGC starts computation on the finest level and uses the coarser levels only to solve correction problems. This works well when a good initial guess on the finest level is available.

Several pieces of this algorithm are major computational issues in their own right. The approximate-solve steps usually use an iterative method on all but possibly the coarsest level. On the coarsest level, a direct solver is frequently used. The approximate-solve steps are frequently referred to in the multigrid community as the *smoothing* steps (see Detour 2, "What Are Smoothers and Roughers?").

Two other steps are the grid transfer steps, namely, interpolation and projection. (For multilevel problems not arising from grids, the corresponding nomenclature is prolongation and restriction.) Both of these steps involve a small amount of the total running time, but are crucial to making multigrid work. See Detour 3, "Grid Transfer Operators."

Nested iteration algorithms. The secondary linear algorithm, a *nested iteration algorithm* called NIC (see Figure 4 for some examples), is defined loosely in Figure 6. NIC is also referred to as *full multigrid* or *FMG*.

Algorithm NIC starts computation on the coarsest level and proceeds to solve problems on each level in order to get an initial guess on the

Glossary

solution space: whatever the solution objects reside in (such as a vector or function space).

smoother: any iterative or direct solver.

rougher: an iterative method that does not necessarily smooth each error component each iteration (for instance, conjugate gradients).

preconditioner: an auxiliary process for accelerating the convergence rate of an iterative solver.

solver: a smoother, a rougher, or a preconditioner for solving a numeric problem.

correction step: a correction obtained from the next coarser grid by solving a related problem (this is actually a preconditioner).

pre-smoother: the solver used before the correction step.

post-smoother: the solver used after the correction step.

prolongation: an interpolation procedure for transferring data from a coarse grid to a fine grid. This can also be a method of mapping a smaller solution space into a larger one.

restriction: a projection method for transferring data from a fine to a coarse grid. This can also be a method of mapping a bigger solution space into a smaller one.

level: this comprises a grid, an associated problem to solve on this grid, a solution space, a (set of) solver(s), and grid transfer methods for transferring data between neighboring levels.

Algorithm MGC(level j, matrices $\{A_i\}$, level j's initial guess x_j^0, right-hand side f_j)
Approximately solve $A_j x_j = f_j$ starting from an initial guess of x_j^0.
If $j > 1$, then possibly do until a condition is met:
  Calculate a residual $r = f_j - A_j x_j$.
  Set $x_{j-1}^0 = 0$ and project r onto f_{j-1}.
  Get update u_j by interpolating the result of MGC$(j-1, \{A_i\}, x_{j-1}^0, f_{j-1})$.
  Approximately solve $A_j x_j = f_j$ starting from an initial guess of $x_j + u_j$.
Return x_j as the approximate solution.

Figure 5. Algorithm MGC, a correction algorithm for linear problems.

Algorithm NIC(finest level k, matrices $\{A_i\}$, level 1's initial guess x_1^0, right-hand sides $\{f_i\}$)
Get x_1 from the result of MGC$(1, \{A_i\}, x_1^0, f_1)$.
If $k > 1$, then do $i = 2, \ldots, k$:
  Get x_i^0 by interpolating x_{i-1}.
  Get x_i from the result of MGC$(i, \{A_\ell\}, x_i^0, f_i)$.
Return x_k as the approximate solution.

Figure 6. Algorithm NIC, a nested iteration algorithm for linear problems.

Algorithm OWMG(finest level k, matrices $\{A_i\}$, level 1's initial guess x_1^0, right-hand sides $\{f_i\}$)
Get x_1 from the result of MGC$(1, \{A_i\}, x_1^0, f_1)$.
If $k > 1$, then do $i = 2, \ldots, k$:
  Get x_i^0 by interpolating x_{i-1}.
  Approximately solve $A_i x_i = f_i$ starting from an initial guess of x_i^0.
Return x_k as the approximate solution.

Figure 7. Algorithm OWMG, another nested iteration algorithm for linear problems.

next finer level until the level k problem is solved. The coarser levels are also used to solve correction problems using Algorithm MGC, hence, the name of the algorithm.

One-way multigrid or OWMG (see Figure 4) corresponds to Algorithm NIC with the main part of Algorithm MGC not performed. Only the initial approximate-solve step is executed, as in Figure 7.

As noted earlier, for problems where there is a good initial guess, Algorithm MGC is a very good choice. If there is no good initial guess, Algorithm NIC will work better than MGC.

There is another fundamental difference between these two algorithms. Suppose the cost of computing the approximate solve and the grid transfers on any level i is CN_i, where $C \in \mathbb{R}$ and N is the number of grid points on level i. To reduce the error on the jth level until it is of the order of the truncation error of the discretization method, Algorithm MGC would normally cost $O(N_j \log N_j)$ while Algorithm NIC would normally cost $O(N_j)$. NIC is faster since when it calls MGC on the non-coarsest levels, the error is already almost of the order of the truncation error. For many problems, the error must be reduced by only a factor on the order of 0.25, which can be done easily.

Linear problems suitable to multilevel approaches come in many well-hidden forms. A good example is how many computer display adapters and displays interact to put an image on a screen. The adapter receives data in the form of a graphics image. This is coarsened using a discrete Fourier transform several times (4 to 8 typically). The much-reduced image is transferred across the connecting wires to the display. The same number of inverse DFTs are applied before the image appears on the screen. This is just a V cycle with one smoother before the coarse grid correction (the DFT) and another smoother after the correction (the inverse DFT). A multilevel wavelet approach can be substituted for the DFT.

Nonlinear problems

Multigrid can be applied to nonlinear problems in two extremely different ways: *directly* using nonlinear iterative methods (for instance, nonlinear Gauss–Seidel) or *indirectly* as part of a linearization technique (for instance, Newton's method).

In some applications one approach is clearly superior. The cost of function evaluations makes the difference. When the function evaluation is inexpensive, the direct application is more cost-effective than computing a Jacobian matrix, which is required by indirect approaches.

All nonlinear multigrid algorithms assume that the original problem has one or more solutions (only one is assumed for theoretical analysis, however). Other assumptions are related to the choices of iterative solvers, solution spaces, and the discretization method.

Direct. Direct applications of multigrid to nonlinear problems are frequently modifications of Algorithm MGC. The most popular of these algorithms is the *full approximation scheme* or *FAS*.[12] It is defined loosely in Figure 8.

When the A_i's are linear operators, this is equivalent to Algorithm MGC. Algorithms FAS and NIC can be combined to get what is called *full multigrid–FAS* or FMG-FAS.

There are many components that can be shared between a linear and a FAS multigrid solver. For example the interpolation and projection routines

Detour 2: What Are Smoothers and Roughers?

The term *smoother*[1] comes from the effect an iterative method has on its error components in Fourier space (see Figure 1). In this space, the components are waves, anywhere from short to long. For a problem like the Poisson equation on a square with a uniform grid, the error analysis can be done entirely with basis functions consisting of sines and cosines. A relaxation method like Gauss–Seidel will reduce the amplitude of every error component by some bounded factor. Hence the components are all "smoothed" by this process.

The ultimate smoothers are direct solvers like (sparse) Gaussian elimination, the fast Fourier transform, or cyclic reduction. However, direct solvers only make sense in multigrid when used on the coarsest grid's problem.

Relaxation methods damp out the high-frequency error components quite quickly while taking a lot longer for low-frequency components. In Figure 1, the error component is represented as a sine curve. On grid 3, it is a low-frequency wave in the sense that it covers eight grid elements. On grid 1, it is a high-frequency wave covering only two elements. On a grid coarser than grid 1, the component would not be representable. A typical relaxation method would dampen this wave out much quicker on grid 1 than grids 2 or 3.

Multigrid methods were motivated by attempts to accelerate (or precondition) relaxation methods. Using the right scale for an error component means the difference between easily damping it out or having to work very hard.

The term *rougher*[2] refers to an iterative method which does not reduce every error component at every iteration. This term is not as prevalent as *smoother*. In fact, many in multigrid refer to all numerical methods, iterative or direct, as smoothers. This is technically incorrect.

In fact, a method like SSOR or conjugate gradients will increase the amplitude of some error components while reducing others. While the error or residual norm is reduced, some components may be severely "roughed" up. This roughing up of error components is standard with iterative methods for nonsymmetric systems of equations.

Many roughers have the property that after a correction step in a multilevel algorithm, the solver amplifies some of the error components that have been essentially obliterated on coarse grids. Most relaxation methods and alternating-direction implicit methods do not do this. Hence, roughers sometimes must recorrect error components on coarser grids that would not reappear if a smoother had been used.

Of course, the choice of whether to use a smoother or a rougher is not entirely trivial. For many problems, roughers provide a better convergence rate for the multigrid algorithm. It comes down to optimizing the convergence rate versus the cost of the iterative procedures.

References

1. A. Brandt, "Multi-Level Adaptive Solutions to Boundary-Value Problems," *Math. of Computation*, Vol. 31, 1977, pp. 333–390.
2. C.C. Douglas and J. Douglas, "A Unified Convergence Theory for Abstract Multigrid or Multilevel Algorithms, Serial and Parallel," *SIAM J. Numerical Analysis*, Vol. 30, 1993, pp. 136–158.

can be the same. The major difference is in the solvers used on each level: specialized relaxation methods for nonlinear problems are usually constructed (see elsewhere[23] for examples). Here, the iterative solvers typically require many function evaluations. However the Jacobian is not formed, which can save much time.

Indirect. Indirect methods are typically modifications of Algorithm NIC. The secondary nonlinear algorithm, a *Newton multigrid* one or NMG, is defined loosely in Figure 9.

The hope in all the Newton multigrid variants (there are several[24,25]) is that as the number of the level increases the number of Jacobian evaluations reduces to one.

Algorithm NMG is really just NIC with a twist. For each level, there are s_j Newton-like iterations where $s_j \to 1$ as $j \to \infty$. The linear correction steps in Algorithm MGC just use the last Jacobian formed on a level.

Once again, many components can be shared between an existing linear and a Newton multigrid solver. For example the interpolation, projection, and solver routines can be the same. Absolutely standard linear equation solvers are used. The additional routines are for the Jacobian evaluation and convergence tests.

Bank and Rose provided the first convergence result for nonlinear multigrid using a damped Newton procedure.[24] The indirect approaches have many theoretical advantages since they are relatively straightforward to analyze. FAS, on the other hand, is notoriously hard to analyze.

Even today at Copper Mountain multigrid conferences, an easy way to start a heated discussion is to declare that application area X (take your choice from semiconductors, reservoir simulation, flame simulation, etc.) is *obviously* best solved by either FAS or a damped Newton multigrid method. Then stand back and listen to the arguments. It can be quite educational.

Algorithm FAS(level k, operators $\{A_i\}$, level k's initial guess x_k^0, level k's right-hand side f_k)
Approximately solve $A_k x_k = f_k$ starting from an initial guess of x_k^0.
If $k > 1$, then possibly do until a condition is met:
  Calculate a residual $r = f_k - A_k x_k$.
  Project x_k onto x_{k-1}^0.
  Set $f_{k-1} = A_{k-1} x_{k-1}^0$ minus the projected r.
  Get u_{k-1} from the result of FAS($k-1$, $\{A_i\}$, x_{k-1}^0, f_{k-1}).
  Get u_k by interpolating $u_{k-1} - x_{k-1}^0$.
  Approximately solve $A_k x_k = f_k$ starting from an initial guess of $x_k + u_k$.
Return x_k as the approximate solution.

Figure 8. Algorithm FAS, a direct algorithm for nonlinear problems.

Algorithm NMG(level k, operators $\{A_i\}$, level 1's initial guess x_1^0, right-hand sides $\{f_i\}$)
Do $i = 1, \ldots, k$:
  If $i > 1$, then get x_i^0 by interpolating x_{i-1}.
  Do $s_i = 1, \ldots$ until convergence is met:
    Form the Jacobian $J_i^{(s_i)}$ using x_i^0.
    Get x_i from the result of MGC(i, $\{J_\ell\}_{\ell=1}^{i-1} \cup J_i^{(s_i)}$, x_i^0, f_i).
    If x_i is not adequate, set $x_i^0 \leftarrow x_i$.
  Set $J_i = J_i^{(s_i)}$.
Return x_k as the approximate solution.

Figure 9. Algorithm NMG, an indirect algorithm for nonlinear problems.

Time-dependent problems

Two basic multigrid approaches can help with time-dependent problems. One applies multigrid directly to well known time-stepping methods. An indirect approach uses multigrid in both time and space.

Multigrid has become a standard tool for solving hyperbolic and evolutionary problems. Since the early 1980s virtually all commercial airplane wings have been designed using multigrid-based numerical simulation.

The direct approach is very simple. At each time step a stationary problem (linear or nonlinear) must be solved. This is done with a standard multigrid method. For parabolic problems, the stationary problem is elliptic. If the convergence rate for the elliptic problem using the multigrid algorithm is some factor $\gamma < 1$, then the convergence rate for the parabolic problem is $C\gamma$, where $C \in \mathbb{R}$ and $C < 1$.

For many applications, however, using multigrid in this manner gives no computational benefit. Some excellent time-extrapolation methods exist which can be coupled to conjugate gradient–like methods. Once the first few time steps are solved (slowly), the extrapolation method provides such a good initial guess to the solution on the next time step that only a few (say, one or two) iterations of the conjugate gradient–like method are needed before moving on. In this situation multigrid cannot compete.

The indirect approach was first investigated in the 1970s by Brandt and his colleagues[26] and more recently by Horton et al.[27] After the first few time steps, the multigrid algorithm does not always go to the finest level before changing the time step. Since the coarse grids have larger mesh spacing, the time steps can be larger too.

For some application areas in which only a stationary solution is wanted, this method works and is extremely fast. However, this does not work for all stationary problems and a number of conditions are required to ensure convergence. For problems in which transient solutions are needed, this method obviously does not apply.

Parallel, telescoping algorithms

The multigrid methods we have considered so far have the property that the number of unknowns per level is reduced as the level number is reduced; that is, $N_i < N_{i+1}$ as in Equation 2. There are two quite distinct families of parallel multigrid algorithms based on maintaining this telescoping property.

Brandt presented the first serious parallel multigrid paper at the 1980 Elliptic Problem Solvers Conference in Santa Fe.[28] He covered the main issues that occur when domain decomposition is used on each level (see Figure 10). While this paper does not have numerical experiments on existing machines, it is just as relevant today as it was then.

There are two ways of combining multigrid and domain decomposition for parallel computers. One is to do multigrid with domain decomposition used to get the parallelism (see Figure 10), which I'll refer to as *MG+DD*. The other is *DD+MG:* first decompose the fine grid using a standard domain decomposition method and then do serial multigrid on each of the subdomain problems (see Figure 11). Multigrid people prefer the former approach. Domain decomposition people prefer the latter.

For simple elliptic PDEs, MG+DD usually takes the same number of multigrid cycles as in the single-processor case. The number of iterations of the smoother may increase a bit, but not which correction cycle is used. For p processors and N unknowns on the finest level, the method is not usually $O(N/p)$ in time, but usually resembles $O((N/p) \log p)$.

On the other hand, DD+MG usually takes a constant number of iterations C of the domain decomposition algorithm (say $C \approx 7$ or 8). The

cost of the multigrid cycle on each subdomain problem takes about as much time as one iteration of MG+DD. Hence, DD+MG usually costs C times as much as MG+DD.

An interesting case study of MG+DD versus DD+MG[29] considers a range of problems from the relatively simple to the extremely difficult. In each case, MG+DD is about 10 times faster than DD+MG. Of course, domain decomposition methods work on complicated domains that multigrid cannot readily be applied to.

In general, though, *when DD+MG works, so does MG+DD and vice versa*. There are exceptions, however. My favorite example is one suggested by Patrick Le Tallec and Jan Mandel. Consider the air system for an office building complex connected by relatively narrow but long hallways. DD+MG models the air flow and temperature distribution beautifully, but MG+DD misses the interesting physics completely.

Why DD+MG works better in this example is illustrative. In DD+MG, the individual buildings provide a natural domain decomposition, one building per subdomain. Getting the physics correct between buildings can be accomplished when solving small problems along the interfaces between the domains (which can include the connecting hallways). MG+DD usually has problems with modeling the hallways owing to differences in scales that tend to lose the hallways.

The other approach to parallel, telescoping multigrid is designed specifically for massively parallel computers. In it, we assume that there is one processor per grid point on all levels. Gannon and Van Rosendale describe such an algorithm for SIMD machines.[30] The same computation occurs on all levels and points simultaneously. Hence, iterative methods like Jacobi or conjugate gradients can be used, but not Gauss–Seidel. The one drawback to this method is that twice as much communication is required as might be expected in order to maintain stability. I describe this method in more detail elsewhere.[31]

Parallel, nontelescoping algorithms

A completely different class of parallel multigrid algorithms keeps the number of unknowns the same on all levels. These were first developed for PDE problems that had odd properties which made it impossible to represent the error on a single coarse grid. Only by having multiple coarse grids could the error be represented and damped out.[32]

Nontelescoping algorithms use two basic approaches. The direct approach is to coarsen

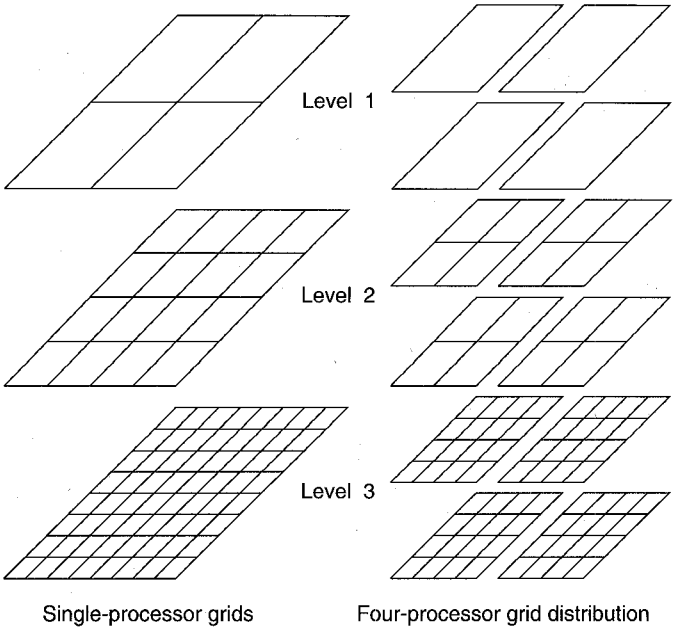

Figure 10. Multigrid, with domain decomposition per level. *Left:* the grids. *Right:* how each grid is decomposed to four processors.

grids by assigning each fine-grid point to one of a set of coarse grids in a regular pattern. The indirect approach is to use the symmetry groups of the PDE to decompose the operator. Both approaches can be done geometrically.

Several nontelescoping multigrid methods

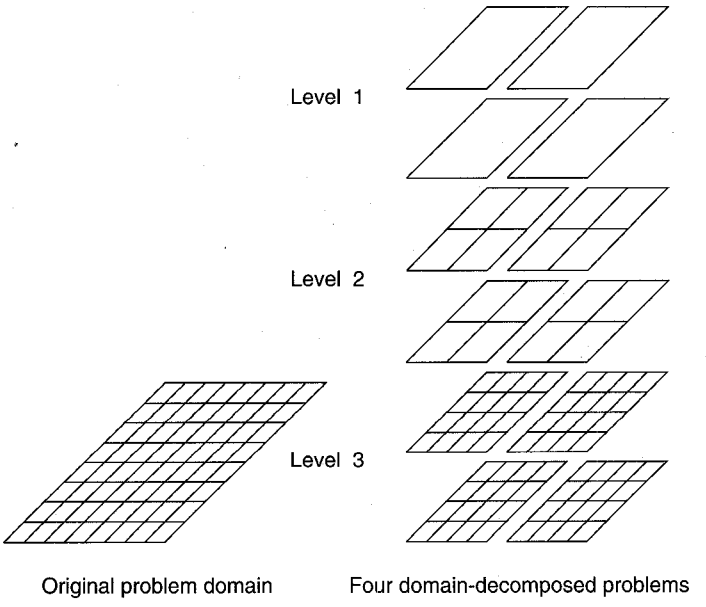

Figure 11. Domain decomposition, with multigrid per subdomain problem. *Left:* the single grid. *Right:* the grids, by level, that each of the four domain-decomposed multigrid problems operates on. Contrast with Figure 10.

Detour 3: Grid Transfer Operators

Transferring information between problems on different grids (or levels) is crucial in making multigrid methods work well.

For simple problems like Poisson's equation on a square, the transfer operators are well understood. When computing the right-hand side for a correction problem, a weighted average of nearby residuals on the finer grid is common. Suppose the residuals in a section of a uniform mesh G_ℓ are numbered as follows:

$$\begin{matrix} r_{i-1,j+1} & r_{i,j+1} & r_{i+1,j+1} \\ r_{i-1,j} & r_{i,j} & r_{i+1,j} \\ r_{i-1,j-1} & r_{i,j-1} & r_{i+1,j-1} \end{matrix}$$

Suppose that the (i, j) point in G_ℓ corresponds to the (i', j') point in grid $G_{\ell-1}$. A common weighting is the discrete $\mathcal{L}_2$ projection, represented by the stencil centered about the point in common between the two grids:

$$f_{i',j'} = C(r_{i-1,j-1} + r_{i+1,j-1} + r_{i-1,j+1} + r_{i+1,j+1} + 2(r_{i,j-1} + r_{i-1,j} + r_{i+1,j} + r_{i,j+1}) + 4r_{i,j}) \quad \text{(A)}$$

where $C \in \{1, 1/16\}$. The choice of constant is determined by the discretization method (it is 1 if the mesh spacing is not factored into the right-hand side).

Equation A can be represented in a *stencil* form where only the multipliers are noted. Recalling Brandt's intergrid transfer notation $I^{\text{to}}_{\text{from}}$ (see Detour 1), I^{2h}_h is a projection (or restriction) operator and I^h_{2h} is an interpolation (or prolongation) operator. This stencil is then commonly written as

$$I^{2h}_h = C \cdot \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}, \quad C \in \{1, \frac{1}{16}\}.$$

Other common projection methods are transposes of discrete interpolation operators. Common interpolation operators are based on linear or cubic interpolation.

An interesting problem in its own right is what is the correct choice for the minimum orders of the projection and interpolation operators for a pth-order partial differential equation. Theory[1] states that the orders may be almost anything as long as the sum of the two is at least $2p - 1$. While violating this sum-of-the-orders rule guarantees disaster normally, this theory does not give a hint as to what the sum of orders does for improving the solution method. For many hard engineering problems, finding a good projection–interpolation pair is the limiting factor on whether or not using multigrid makes sense.

Consider a typical flame. Near the flame front, large changes occur in the temperature, velocities, and information about the chemical species. Though the computed solutions are continuous, they behave as if they are jumping across the flame front. While adaptively chosen grids reduce the size of these jumps, there is still a noticeable jump on reasonably sized grids. In fact, very few computers have enough memory to reduce the jumps adequately. Because of these jumps, the nonlinear multigrid methods used in this problem do not always compute an acceptable correction on a coarser grid. Finding better projection–interpolation pairs alleviates this problem.

Reference

1. P.W. Hemker, "On the Order of Prolongations and Restrictions in Multigrid Procedures," *J. Computational and Applied Math.*, Vol. 32, 1990, pp. 423–429.

have been explored. The three most common are those by Brandt and Ta'asan, Frederickson and McBryan, and Hackbusch. For oddly shaped domains, Dendy has spent years arguing that Brandt and Ta'asan's method is superior to the other two, particularly from an implementation viewpoint. From a convergence viewpoint, McBryan has demonstrated that his method is superior to the other two methods (assuming all three converge).

The indirect approach uses the fact that the domain can be folded in particular ways related to the symmetry groups of the PDE. This leads to two-level methods with possibly a great many subproblems[33] (8 for a square, 64 to 192 for a cube). The subproblems are usually constructed so as to be mutually orthogonal under the natural inner product for the original problem. Papers have been written about this topic by me and a collection of coauthors, principally Miranker, Mandel, and Barry Smith.

In addition, these problems are themselves boundary value problems with predictable boundary conditions. Assuming all of the boundary conditions were Dirichlet originally, a combination of Dirichlet and Neumann conditions result in the subproblems. Other, simple boundary conditions can be handled automatically.

An interesting fact is that many of the subproblems are similar to each other except for the boundary conditions. A sparse matrix solver was devised that solved problems of the form $(A_j + B_i)x^i_j = f^i_j$, where A_j is sparse. B_i is super-sparse and only corrects for rows associated with boundary points in the grid to reflect the correct boundary condition there.

Detour 4: Structured Spaghetti Code Can Win Big

Multigrid for PDEs consists of four major components:

(1) Solvers (for example, relaxation methods)
(2) Residual computation
(3) Interpolation
(4) Projection (for example, weighted average of residuals)

Each of these is usually coded separately, providing a well-structured code in which the pieces can easily be replaced with a different algorithm.

A different approach is to write highly structured codes that implement a part of each of these components assuming that there is a well-defined loop structure over the domain. Most computer languages used by scientists and engineers have a construct for including code from other files into a program. Using this technique, all four of the components can be combined into a much more complicated routine. However, what has to be written is not really any harder to code than the usual programs.

By slightly tuning one or two parameters that describe how many cache lines there are and the size of each, data in cache can be reused many times. In fact, if enough effort is made, data passes through cache only once before changing levels.[1] If data would normally pass through cache m times, this can lead to a speedup of almost $m-1$ times since moving data to and from main memory tends to determine the actual speed of most algorithms on RISC-based machines.

Reference

1. C.C. Douglas, "Caching In with Multigrid Algorithms: Problems in Two Dimensions," *Parallel Algorithms and Applications*, Vol. 9, 1996, pp. 195–204.

This was devised to take advantage of the fact that the rows of B_i are almost entirely all 0's. Doing this allows for a parallel multilevel solver that uses much less storage than a conventional iterative or multigrid procedure. Elsewhere I explain the details.[33]

An extensive description of both styles of parallel, nontelescoping multigrid, along with a list of citations, appears elsewhere.[31]

Studies by the National Science Foundation show that while improvements in hardware over the past 20 years have contributed to a several-order-of-magnitude speedup in solving problems, improvements in algorithms have helped even more. Multigrid is one of the principal algorithmic improvements cited.

Multigrid continues to become the standard iterative method of choice to replace sparse Gaussian elimination and to precondition traditional iterative methods in more and more problems in the sciences and engineering.

As we approach the year 2000, several research areas will be quite active. One is determining how to coarsen unstructured grids for complicated problems. Doing this blindly leads to barely convergent methods, or ones that do not use the CPU's cache reasonably. (See Detour 4 for a discussion of caches in multigrid.) Applications to other areas (such as sparse matrix methods in general) will be a side benefit.

Another active research area is in solving complicated nonlinear, possibly time-dependent problems. Several algorithms must interact: time stepping, nonlinear algorithms, multigrid algorithms, and linear solvers. We don't know yet how to optimize how well each algorithm must solve its part of the problem. Even predetermining that the algorithms will work together successfully is often an open question. Progress in this area will have a major impact on many fields that affect all of us daily. ♦

References

1. W.L. Briggs, *A Multigrid Tutorial*, SIAM, Philadelphia, 1987.
2. W. Hackbusch, *Multigrid Methods and Applications*, Springer Series in Computational Mathematics Vol. 4, Springer-Verlag, Berlin, 1985.
3. P. Wesseling, *An Introduction to Multigrid Methods*, John Wiley & Sons, Chichester, UK, 1992.
4. J.H. Bramble, *Multigrid Methods*, Pitman Research Notes in Mathematical Sciences Vol. 294, Longman Scientific & Technical, Essex, UK, 1993.
5. B.F. Smith, P.E. Bjørstad, and W.D. Gropp, *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*, Cambridge Univ. Press, New York, 1996.
6. R.V. Southwell, *Relaxation Methods in Engineering Science*, Oxford Univ. Press, Oxford, UK, 1940.
7. R.P. Fedorenko, "A Relaxation Method for Solving Elliptic Difference Equations," *Zhurnal vychislitel'noj matemaki i matematicheskoj fiziki*, Vol. 1, 1961, pp. 922–927. Also in *USSR Computational Math. and Math. Physics*, Vol. 1, 1962, pp. 1092–1096.
8. R.P. Fedorenko, "The Speed of Convergence of One Iteration Process," *Zhurnal vychislitel'noj matemaki i matematicheskoj fiziki*, Vol. 4, 1964,

pp. 559–563. Also in *USSR Computational Math. and Math. Physics*, Vol. 4, 1964, pp. 227–235.
9. N.S. Bakhvalov, "On the Convergence of a Relaxation Method Under Natural Constraints on an Elliptic Operator," *Zhurnal vychislitel'noj matemaki i matematicheskoj fiziki*, Vol. 6, 1966, pp. 861–883.
10. G.P. Astrakhantsev, "An Iterative Method of Solving Elliptic Net Problems," *Zhurnal vychislitel'noj matemaki i matematicheskoj fiziki*, Vol. 11, 1971, pp. 439–448.
11. R.E. Bank and T. Dupont, "An Optimal Order Process for Solving Elliptic Finite Element Equations," *Math. of Computation*, Vol. 36, 1981, pp. 35–51.
12. A. Brandt, "Multi-Level Adaptive Solutions to Boundary-Value Problems," *Math. of Computation*, Vol. 31, 1977, pp. 333–390.
13. C.C. Douglas, "Multi-Grid Algorithms with Applications to Elliptic Boundary-Value Problems," *SIAM J. Numerical Analysis*, Vol. 21, 1984, pp. 236–254.
14. R.E. Bank and C.C. Douglas, "Sharp Estimates for Multigrid Rates of Convergence with General Smoothing and Acceleration," *SIAM J. Numerical Analysis*, Vol. 22, 1985, pp. 617–633.
15. R.E. Bank, T. Dupont, and H. Yserentant, "The Hierarchical Basis Multigrid Method," *Numerische Mathematik*, Vol. 52, 1988, pp. 427–458.
16. J.W. Ruge and K. Stüben, "Algebraic Multigrid (AMG)," in *Multigrid Methods*, S.F. McCormick, ed., Frontiers in Applied Mathematics Vol. 3, SIAM, Philadelphia, 1987, pp. 73–130.
17. O. Axelsson and P.S. Vassilevski, "Algebraic Multilevel Preconditioning Methods, I," *Numerische Mathematik*, Vol. 56, 1989, pp. 157–177.
18. O. Axelsson and B. Polman, eds., *AMLI '96: Proc. Conf. on Algebraic Multilevel Iteration Methods with Applications*, Univ. of Nijmegan, Nijmegan, The Netherlands, 1996.
19. C.C. Douglas and M.B. Douglas, "MGNet Bibliography," Yale University, New Haven, Conn., 1991–96. Available on MGNet at http://casper.cs.yale.edu/mgnet/www/mgnet.html.
20. J.H. Bramble et al., "Convergence Estimates for Multigrid Algorithms without Regularity Assumptions," *Math. of Computation*, Vol. 57, 1991, pp. 23–45.
21. W.F. Mitchell, *Unified Multilevel Adaptive Finite Element Methods for Elliptic Problems*, doctoral dissertation, Univ. of Illinois at Urbana-Champaign, Urbana, Ill., 1988. (Also available on MGNet.)
22. U. Rüde, *Mathematical and Computational Techniques for Multilevel Adaptive Methods*, Frontiers in Applied Mathematics Vol. 13, SIAM, Philadelphia, 1993.
23. K. Stüben and U. Trottenberg, "Multigrid Methods: Fundamental Algorithms, Model Problem Analysis and Applications," in *Multigrid Methods*, W. Hackbusch and U. Trottenberg, eds., Lecture Notes in Mathematics Vol. 960, Springer-Verlag, Berlin, 1982, pp. 1–176.
24. R.E. Bank and D.J. Rose, "Analysis of a Multilevel Iterative Method for Nonlinear Finite Element Equations," *Math. of Computation*, Vol. 39, 1982, pp. 453–465.
25. W. Hackbusch and A. Reusken, "Analysis of a Damped Nonlinear Multilevel Method," *Numerische Mathematik*, Vol. 55, 1989, pp. 225–246.
26. A. Brandt and N. Dinar, "Multigrid Solutions to Elliptic Flow Problems," in *Numerical Methods for Partial Differential Equations*, S. Parter, ed., Academic Press, New York, 1979, pp. 53–147.
27. G. Horton, "The Time-Parallel Multigrid Method," *Comm. Applied Numerical Methods*, Vol. 8, 1992, pp. 585–595.
28. A. Brandt, "Multigrid Solvers on Parallel Computers," in *Elliptic Problem Solvers*, M.H. Schultz, ed., Academic Press, New York, 1981, pp. 39–83.
29. U. Gärtel and K. Ressel, "Parallel Multigrid: Grid Partitioning Versus Domain Decomposition," *Proc. 10th Int'l Conf. Computing Methods in Applied Sciences and Engineering*, R. Glowinski, ed., Nova Science Publishers, New York, 1992, pp. 559–568.
30. D. Gannon and J.R. Rosendale, "Highly Parallel Multigrid Solvers for Elliptic PDEs: An Experimental Analysis," Tech. Report 82-36, ICASE, Hampton, Va., 1982.
31. C.C. Douglas, "A Review of Numerous Parallel Multigrid Methods," in *Applications on Advanced Architecture Computers*, G. Astfalk, ed., SIAM, Philadelphia, 1996, pp. 187–202.
32. A. Brandt and S. Ta'asan, "Multigrid Methods for Nearly Singular and Slightly Indefinite Problems," in *Multigrid Methods II*, W. Hackbusch and U. Trottenberg, eds., Springer-Verlag, Berlin, 1986, pp. 99–121.
33. C.C. Douglas, "A Tupleware Approach to Domain Decomposition Methods," *Applied Numerical Math.*, Vol. 8, 1991, pp. 353–373.

Craig C. Douglas *can be found most often in an airplane, but has also been sighted at IBM's T.J. Watson Research Center (formerly), and Yale University's Department of Computer Science. He recently moved to the University of Kentucky. From late 1993 through 1995 he was codirector of the Algorithms group at CERFACS (Toulouse, France). Douglas received an AB in mathematics from the University of Chicago and a MS, MPhil, and PhD in computer science from Yale. He is managing editor of MGNet and an organizer of the Copper Mountain conferences, and regularly publishes papers on multigrid and domain decomposition methods, parallel algorithms, linear algebra, and sparse matrix methods. Douglas is a member of SIAM, ACM, and the IEEE Computer Society.*

Douglas can still be reached at 8 South Street, Cos Cob, CT 06807-1618, USA; e-mail, cdouglas@na-net.ornl.gov; WWW, http://www.ms.uky.edu/~ccd.