RIACS

Research Institute for Advanced Computer Science
NASA Ames Research Center

A Highly Parallel Multigrid-Like Method for the Solution of the Euler Equations

Ray S. Tuminaro

Research Institute for Advanced Computer Science
NASA Ames Research Center - MS: 230-5
Moffett Field, CA 94035

RIACS Technical Report 89.59

December 1989

The Research Institute of Advanced Computer Science is operated by Universities Space Research Association, The American City Building, Suite 311, Columbia, MD 244, (301)730-2656

Work reported herein was supported in part by Cooperative Agreements NCC 2-387 between the National Aeronautics and Space Administration (NASA) and the Universities Space Research Association (USRA).

A Highly Parallel Multigrid-Like Method for the Solution of the Euler Equations

Ray S. Tuminaro

December 1989

Research Institute for Advanced Computer Science
NASA Ames Research Center

RIACS Technical Report 89.59

NASA Cooperative Agreement Number NCC 2-387

A Highly Parallel Multigrid-like Method for the Solution of the Euler Equations

Ray S. Tuminaro*

Research Institute for Advanced Computer Science, NASA Ames

Abstract. We consider a highly parallel multigrid-like method for the solution of the two dimensional steady Euler equations. The new method, introduced in [4] as "filtering" multigrid, is similar to a standard multigrid scheme in that convergence on the finest grid is accelerated by iterations on coarser grids. In the filtering method, however, additional fine grid subproblems are processed concurrently with coarse grid computations to further accelerate convergence. These additional problems are obtained by splitting the residual into a smooth and an oscillatory component. The smooth component is then used to form a coarse grid problem (similar to standard multigrid) while the oscillatory component is used for a fine grid subproblem. The primary advantage in the filtering approach is that fewer iterations are required and that most of the additional work per iteration can be performed in parallel with the standard coarse grid computations.

In this paper, we generalize the filtering algorithm to a version suitable for nonlinear problems. We emphasize that this generalization is conceptually straight-forward and relatively easy to implement. In particular, no explicit linearization (e.g. formation of Jacobians) needs to be performed (similar to the FAS multigrid approach). We illustrate the nonlinear version by applying it to the Euler equations, and presenting numerical results. Finally, a performance evaluation is made based on execution time models and convergence information obtained from numerical experiments.

1. Introduction. Multigrid methods are among the fastest algorithms for a wide variety of problems and are now used in many scientific disciplines. Structurally, the algorithm iterates on a hierarchy of consecutively coarser and coarser grids until convergence

* This work performed at Research Institute for Advanced Computer Science (RIACS) and was supported under Cooperative Agreement NCC 2-387 between NASA and the Universities Space Research Association (USRA).

is reached. While critical to its rapid convergence, the coarse grid computations are more difficult to parallelize efficiently due to the presence of fewer grid points (and hence less parallelizable work). We therefore consider a highly parallel multigrid-like method (see [5,6,7] for other types of highly parallel multigrid-like methods). This new algorithm, "filtering" (proposed in [4]), uses additional fine grid subproblems to accelerate the convergence of the overall process. More specifically, these fine grid problems are created by splitting the residual into a smooth and an oscillatory component. The smooth component is used to form a coarse grid problem (similar to standard multigrid) while the oscillatory component is used for the fine grid subproblem. The primary benefit to this approach is that while more work per iteration is necessary, fewer iterations are required and more of the work within an iteration is parallelizable. In fact, if the additional work can be performed concurrently with coarse grid computations, the CPU time per iteration need not rise significantly.

In this paper, we generalize the filtering algorithm into a version suitable for nonlinear problems. This new algorithm is conceptually straight-forward and relatively easy to implement. In particular, no explicit linearization (e.g. formation of Jacobians) needs to be performed (similar to the FAS multigrid approach). We apply the nonlinear version to the solution of the Euler equations (see [4,10] for convergence analysis of the filtering algorithm for linear model problems). Specifically, we consider the filtering approach applied to the FLO52 algorithm. FLO52, written by Antony Jameson [8], is a well-known multigrid code for the solution of the Euler equations describing transonic flow past an airfoil. The corresponding FLO52-Filtering algorithm is similar to the original FLO52 with the exception of the additional subproblems. We begin our description of the algorithm with discussions of both the standard and filtering multigrid methods applied to linear problems in Section 2, and Section 3. The generalization of both the standard multigrid and the filtering approach to nonlinear problems is then discussed in Section 4 and Section 5. We conclude by comparing the convergence of the filtering and standard FLO52 algorithms on a fluid calculation. Based on these numerical experiments as well as a mathematical execution time model, we make some predictions on the performance of the FLO52-Filtering algorithm on massively parallel computers.

2. Standard Multigrid Algorithm. We begin our discussion with a brief sketch of the standard multigrid algorithm applied to linear elliptic partial differential equations

(PDE's). More complete introductory material on multigrid methods can be found in [3,9].

Assume that a given elliptic partial differential equation is approximated by a discrete set of equations (finite differences or finite elements):

$$A_1 u = b, \tag{1}$$

where A_1 is a matrix, b is a vector, and u is a vector of unknowns for which we seek the solution. One iteration of a simple multigrid ('V' cycle) method consists of the following steps:

- relaxation iterations (e.g. Jacobi or SOR methods),
- formation of a correction equation for the error in the current approximation,
- projection of correction equation onto a coarser grid,
- 'solution' of coarse grid system,
- interpolation and addition of correction to previous approximation.

A key feature of this procedure is that the solution to the coarse grid equations can be approximated using the multigrid idea recursively. Thus, the general algorithm consists of processing on a hierarchy of coarser grids (each processed in turn). We summarize this multigrid algorithm with a pseudo-code fragment in Fig. 1.

For the most part, analysis of the two-level multigrid method reveals the general behavior of the multiple grid version. If we denote the projection operator by R_1, the interpolation operator by P_1, the relaxation iteration operator by G, and the coarse grid difference operator by A_2, we can express the two-grid iteration operator,

$$e^{(k+1)} = T e^{(k)}, \tag{2}$$

by

$$T = (A_1^{-1} - P_1 A_2^{-1} R_1) A_1 G, \tag{3}$$

where $e^{(k)}$ denotes the error after k iterations. In the next section, we will contrast this two-grid operator with that of the two-level filtering method.

3. Filtering Algorithm. Conceptually, the filtering algorithm is similar to the standard multigrid method, the primary difference being that two correction equations are

```
proc Multigrid(A_i, b, u, level)
{
    if ( level = CoarsestLevel ) then u = A_i^{-1} b
    else
        PreRelax( A_i, b, u,level)
        ComputeResidual(b, u, level,residual)
        ProjectResidual(level,residual,coarse_residual)
        Multigrid(A_{i+1},coarse_residual,v,level+1)
        Interpolate(level,v,correction)
        u = u+correction
    endif
}
```

. *Simple 'V' cycle multigrid algorithm.*

formed after the relaxation iterations. Specifically, let u_1 denote an approximate solution to the linear equation

$$A_1 u = b. \tag{4}$$

Within a standard multigrid method, a correction equation is usually formed by computing the residual ($r = b - A_1 u_1$) and using this as the righthand side to a new equation. Within the filtering algorithm, however, two subproblems are created by splitting the residual into the two components:

$$r_1 = Zr \quad \text{and} \quad r_2 = r - r_1 \tag{5}$$

which are then used as righthand sides in

$$A_1 c^{(1)} = r_1 \quad \text{and} \quad A_1 c^{(2)} = r_2. \tag{6}$$

To approximate the solution of the first subproblem, the equation is projected onto a coarser grid as within a standard multigrid method. The error associated with this approximation (solving the coarse problem exactly) is

$$e_1 = (A_1^{-1} - P_1 A_2^{-1} R_1) r_1, \tag{7}$$

```
proc FilterMultigrid( A_i, b, u, level )
 if ( level = CoarsestLevel ) then u = A_i^{-1} b;
 else
      PreRelax(b, u,level);
      ComputeResidual(A_i, u, b, level, res);
      SplitResidual(res, r_1, r_2 );
      ProjRes(r_1, r̂_1);
      FilterMultigrid( A_{i+1}, r̂_1, û_1, level+1);   ConcurrentRelax(A_i, r_2, u_2, level+1);
      Interpolate_and_add(û_1, u_2, level, u );
 endif
```

. *V cycle version of the filtering algorithm.*

where A_2 is the coarse grid operator. To approximate the solution of the second problem, q relaxation sweeps are performed on the fine grid. The error for this second approximation is given by

$$e_2 = S^q A_1^{-1} r_2, \tag{8}$$

where S is the iteration operator of the concurrent relaxation method and we have assumed that the initial guess is zero. Thus, the two level algorithm generates one coarse grid correction (as in standard multigrid method) and one additional fine grid problem to be processed by relaxation iterations. Once again, a multiple grid version of the method can be defined by recursively using the filtering procedure to 'solve' the coarse grid subproblems. This multilevel version is summarized in Fig. 2.

Finally, an iteration operator for the two-level version of this algorithm is obtained by combining (7) and (8):

$$e^{(k+1)} = [(A_1^{-1} - P_1 A_2^{-1} R_1)Z + S^q A_1^{-1}(I - Z)]A_1 G e^{(k)}, \tag{9}$$

where we have included the possibility of performing one prerelaxation sweep with iteration operator G. That is,

$$r = A_1 G e^{(k)}. \tag{10}$$

Notice that when the operator Z is the identity matrix, (9) is identical to (3).

q	$\rho(T_f)$
1.	.498
2.	.268
3.	.230
4.	.205

Convergence rates of filtering algorithm corresponding to a 32×32 *grid.*

Intuitively, the S^q term in (9) damps the high frequencies while the coarse grid correction damps the low frequencies. A critical element affecting the convergence behavior of the filtering algorithm is the properties of the splitting operator, Z, used for computing r_1 and r_2. For the most part, this operator should decompose the residual so that high frequency errors remain on the fine grid while low frequency errors are projected on to the coarse grid subproblem. This can be accomplished by choosing an operator which filters out high frequencies in the residual. Many choices are possible. In this paper, we consider only the operator

$$Z = P_1 R_1. \tag{11}$$

This corresponds to first projecting the residual onto the coarse grid and then interpolating back to the fine grid. See [4] for an alternative filter.

A detailed convergence analysis (via. Fourier transform on the iteration operator) of a two-level filtering algorithm applied to the Poisson equation can be found in [4,10]. Using this analysis, it is possible to determine convergence rates for the filtering method. Table 1 lists the spectral radius of the two-grid iteration operator as a function of q when the algorithm is applied to the Poisson equation:

$$u_{xx} + u_{yy} = f. \tag{12}$$

The algorithm depicted uses full-weighted restriction, given by the stencil

$$\frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}, \tag{13}$$

bilinear interpolation, one Jacobi prerelaxation sweep, q concurrent relaxation iterations of damped-Jacobi with damping parameter equal to 4/5, and the exact solution of the

coarse grid equations. Finally, discretization is obtained via central differences on both the fine grid (32×32) and the coarse grid (16×16). We note that by comparison, the corresponding standard multigrid method using one damped Jacobi relaxation sweep (with optimal damping parameter) has a spectral radius of .570. Thus even with only one concurrent relaxation sweep, the convergence rate is accelerated.

4. FAS-Multigrid Method. To apply the multigrid method to a nonlinear problem, the simple scheme described in Section 2 must be modified to implement the FAS algorithm. For the most part, these modifications ensure that the correction equations correspond to physically meaningful subproblems. To describe the algorithm, we consider the nonlinear systems

$$A_k(u) = f_k \tag{14}$$

arising from discretization of a partial differential equation on grid $\mathcal{G}_k$ (where $k = 0$ corresponds to the finest grid). We write one iteration of the relaxation scheme $\mathcal{G}_k$ as

$$u_k \leftarrow Relax(u_k, f_k). \tag{15}$$

Once again, let R_k denote projection from grid k to grid $k+1$. Similarly, let P_k denote interpolation from grid k to grid $k-1$. Then the coarse grid subproblem is defined as follows. The initial guess on the coarse grid is given by

$$u_{k+1} \leftarrow R_k u_k. \tag{16}$$

On the finest grid f_0 corresponds to the discretization of the continuous righthand side. The righthand sides on the coarser meshes are recursively defined by

$$f_{k+1} \leftarrow A_{k+1}(u_{k+1}) - R_k r_k, \tag{17}$$

where

$$r_k \leftarrow f_k - A_k(u_k). \tag{18}$$

Finally, after the solution on the coarse grid is improved (either by relaxation or recursively applying the multigrid procedure) the solution on the fine grid is corrected by

$$u_k \leftarrow u_k + P_{k+1}(u_{k+1} - R_k u_k). \tag{19}$$

Below we summarize a two-grid version of the algorithm using one relaxation sweep on each grid level:

$$\begin{aligned} u_0 &\leftarrow Relax(u_0, f_0) \\ u_1 &\leftarrow R_0 u_0 \\ r_0 &\leftarrow f_1 - A_1(u_1) \\ f_1 &\leftarrow A_1(u_1) - R_0 r_0 \\ u_1 &\leftarrow Relax(u_1, f_1) \\ u_0 &\leftarrow u_0 + P_1(u_1 - R_0 u_0). \end{aligned}$$

For more than two grids, the algorithm is recursively defined by replacing

$$u_1 \leftarrow Relax(u_1, f_1)$$

with

$$u_1 = \text{result of FAS algorithm starting on } \mathcal{G}_1.$$

Notice that when this method is applied to a linear problem, it is mathematically identical to the method described in Section 2. See [2] for more on the FAS procedure.

5. FAS-Filtering Algorithm. Similar to the FAS algorithm, the filtering algorithm must be modified so that the all subproblems are physically meaningful. To describe the method, we consider the discrete nonlinear systems defined by (14):

$$A_k(u) = f_k. \tag{20}$$

Similar to the FAS scheme, a coarse grid subproblem is created after relaxation on $\mathcal{G}_k$ by first computing the residual on that level. However, in the filtering version this residual is further split into two components and then the smooth component is used in forming the coarse grid subproblem. That is,

$$r_k = f_k - A_k(u_k), \quad \hat{r}_k = Z r_k, \quad \text{and} \quad \bar{r}_k = r_k - \hat{r}_k. \tag{21}$$

The formation of the coarse grid subproblem proceeds in a similar fashion to that of a standard FAS algorithm with the exception that $\hat{r}_k$ is used instead of r_k. That is, the initial guess on $\mathcal{G}_{k+1}$ is

$$u_{k+1} = R_k u_k, \tag{22}$$

and the righthand side of the coarse grid equations are recursively defined by:

$$f_{k+1} = A_{k+1}(u_{k+1}) - R_k \hat{r}_k. \tag{23}$$

In addition to the coarse grid subproblem, a fine grid problem is created

$$A_k(\tilde{u}_k) = \tilde{f}_k, \tag{24}$$

which can be processed concurrently with the coarse grid problem. The righthand side of this fine grid subproblem is defined by

$$\tilde{f}_k = A_k(u_k) + \tilde{r}_k, \tag{25}$$

and the initial guess to this system is taken to be u_k. Similar to the filtering algorithm described in Section 3, a relaxation scheme is used to improve the approximation to (24). Finally, after the approximations to the coarse grid and fine grid subproblems have been improved, the solution of the original problem is corrected by

$$u_k \leftarrow u_k + P_{k+1}(u_{k+1} - R_k u_k) + (\tilde{u}_k - u_k). \tag{26}$$

Below we summarize the two-grid version of the algorithm using one prerelaxation sweep, one concurrent relaxation sweep on the fine grid, and 1 relaxation sweep on the coarse grid:

$$\begin{aligned}
u_0 &\leftarrow Relax(u_0, f_0) \\
r_0 &\leftarrow f_0 - A_0(u_0) \\
\hat{r}_0 \leftarrow Z r_0 \qquad & \qquad \tilde{u}_0 \leftarrow u_0 \\
u_1 \leftarrow R_0 u_0 \qquad & \qquad \tilde{r}_0 \leftarrow r_0 - \hat{r}_0 \\
f_1 \leftarrow A_1(u_1) - R_0 \hat{r}_0 \qquad & \qquad \tilde{f}_0 \leftarrow A_0(\tilde{u}_0) + \tilde{r}_0 \\
u_1 \leftarrow Relax(u_1, f_1) \qquad & \qquad \tilde{u}_0 \leftarrow Relax(\tilde{u}_0, \tilde{f}_0) \\
u_0 &\leftarrow u_0 + P_1(u_1 - R_0 u_0) + (\tilde{u}_0 - u_0).
\end{aligned}$$

In the above pseudo-code fragment, independent parts of the two subproblems appear in separate columns. A multilevel version of the above algorithm can be obtained by replacing

$$u_1 \leftarrow Relax(u_1, f_1)$$

with

$$u_1 = \text{result of FAS-Filtering algorithm starting on } \mathcal{G}_1.$$

One can easily verify that when the operators A_i are linear, the FAS-Filtering method is mathematically identical to that described in Section 3.

It is important to realize the relative simplicity of modifying an FAS method to implement the FAS-Filtering algorithm. One advantage is that no linearization is needed. Only the splitting operator and the concurrent relaxation operator must now be developed. For the concurrent relaxation one can use the same (or a similar) routine as for the prerelaxation. Additionally, no additional work is required to implement the splitting operator if

$$Z = P_k R_k$$

is used as these operators are already defined for the interpolation and restriction. In fact, the only aspect of the filtering algorithm that requires nontrivial modification are the routines necessary for the allocation of subproblems to different processors.

6. FLO52 and the Euler Equations. We consider both the FAS and the filtering schemes applied to the Euler equations. We begin by describing the Euler equations and the FLO52 code.

The FLO52 algorithm written by Antony Jameson solves the two-dimensional steady Euler equations describing flow around an airfoil. It is widely used in research and industrial applications throughout the world. It produces good results for problems in its domain of application (steady inviscid flow around a two-dimensional body), and converges rapidly.

We briefly describe the Euler equations and the FLO52 scheme (see [8] for more on FLO52). We begin with the unsteady time-dependent two-dimensional equations written in conservation integral form as

$$\frac{d}{dt}\iint w + \oint \mathbf{n}\cdot\mathbf{F} = 0 \tag{27}$$

where $\mathbf{n}$ is the outward pointing normal on the boundary of the region. The variable w is the vector of unknowns

$$w = (\rho, \rho u, \rho v, \rho E)^T, \tag{28}$$

where ρ is density, u and v are velocity components directed along the x and y-axes, respectively, and E is total energy per unit mass. The function $\mathbf{F}$ is given by

$$\mathbf{F}(w) \leftarrow (E(w), F(w)), \tag{29}$$

where

$$\begin{aligned} E(w) &= (\rho u, \rho u^2 + p, \rho uv, \rho uH)^T, \\ F(w) &= (\rho v, \rho uv, \rho v^2 + p, \rho vH)^T. \end{aligned}$$

Here p is pressure, and H is enthalpy. These are defined by

$$\begin{aligned} p &= (\gamma - 1)\rho[E - (u^2 + v^2)/2], \\ H &= E + p/\rho, \end{aligned}$$

where γ is the ratio of specific heats. The integral relation given by (27) expresses conservation of mass, momentum, and energy which is to hold for any region in the flow domain.

To produce a numerical method based on (27), the flow domain is divided into quadrilaterals. On each quadrilateral of the domain, the double integral in (27) is approximated by the centroid rule and the line integral is approximated by the midpoint rule. For numerical stability, a dissipation term which is a blend of second and fourth-order differences is added.

A simple iterative method (such as the Jacobi algorithm) for the steady-state problem can be viewed as a time-marching method for the time-dependent equations (27). After spatial discretization, the equations form a system of ordinary differential equations

$$\frac{dw}{dt} + A_1(w) = 0, \tag{30}$$

where $A_1()$ denotes the nonlinear finite-difference operator corresponding to differencing of spatial derivatives. Thus for the steady-state solution, we are interested in solving

$$A_1(w) = 0. \tag{31}$$

The application of both the FAS multigrid and filtering algorithms to this problem is relatively straight-forward. We conclude this section with a description of the relaxation

method used within the algorithm. Specifically, it is a general multistage Runge-Kutta-like method. Such a procedure can be written in the form

$$
\begin{aligned}
w^{(0)} &= w_n \\
\text{for } k &= 1 \text{ to } m \\
& \qquad w^{(k)} = w^{(0)} - \Delta t \sum_{j=0}^{k-1} \alpha_{kj} A_i(w^{(j)}) \\
w_{n+1} &= w^{(m)}
\end{aligned}
\tag{32}
$$

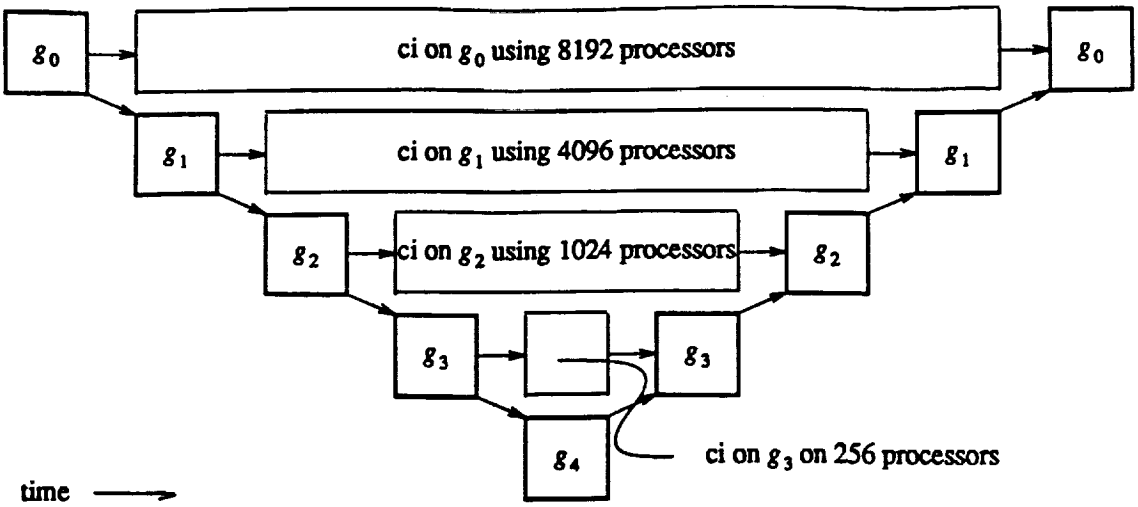
where Δt, the time step, and α_{kj} are chosen so that the procedure converges rapidly. Finally, we note that a number of acceleration techniques are used to improve this Runge-Kutta scheme [1,8] which we do not discuss in this paper.

7. Algorithmic Choices and Operation Counts. To compare the performance of the filtering approach to a standard FAS multigrid procedure, we modified the FLO52 algorithm to implement the procedure described in the previous section (denoted FLO52-Filtering) on a serial machine. Using these codes, we compare the number of iterations required for convergence using FLO52 and FLO52-Filtering. Since the code was not implemented on a parallel machine, operation counts are made for both algorithms so as to compare the time per iteration of both approaches. Additionally, the operation counts were used to determine the number of concurrent iterations that can be performed in parallel with the coarse grid correction. In this section, we describe the algorithmic choices and briefly discuss the corresponding floating point operation counts for each of the FLO52 algorithms.

The operators used in our experiments are those that are typically used within the FLO52 code. Specifically, the Runge-Kutta relaxation scheme described in the previous section is used for prerelaxation and concurrent relaxation. Bilinear interpolation, and full-weighted restriction are used to transfer values between grids. The coarse grid contains one fourth as many points as the fine mesh. Coarse grid operators are defined using the same discretization scheme as on the fine grid. Finally, as discussed in Section 3, the operator Z which splits the residual in the filtering algorithm is given by

$$Z = P_k R_k. \tag{33}$$

This corresponds to projecting the residual onto the coarse grid and then interpolating it to the fine grid.



. *Illustration of the subproblems (and processor allocation) for a 5-level filtering method.*

To estimate the time per iteration of the FLO52 and FLO52-Filtering algorithms, we make a number of assumptions. The first is that the time for communication is negligible.[1] The finest grid contains 256 × 64 points and that each grid point is assigned to one processor. This implies that many processors are inactive when processing coarser meshes in the FLO52 procedure. In particular, each successively coarser grid contains one fourth as many points as the previous grid. Thus, the number of idle processors when processing on $\mathcal{G}_k$ is given by

$$P(1 - \frac{1}{4}^k), \tag{34}$$

where P is the total number of processors. For the FLO52-Filtering algorithm, we assume that these inactive processors are used for the concurrent iterations. By way of example, Fig. 3 illustrates the different subproblems in a FLO52-Filtering algorithm using 5 grid levels. In the figure, the darker boxes correspond to the standard multigrid 'V' cycle. Within each box compromising the 'V' cycle, the grid on which processing occurs is indicated. Each element of the first half of the 'V' cycle (except for the coarsest grid) spawns a concurrent iteration problem. This is indicated by the lighter boxes which are

[1] Since the two multigrid schemes that we compare have similar communication requirements, the relative performance of the two algorithms should not be highly sensitive to the communication times.

μ	Floating Point Operations	
	FLO52	FLO52-Filtering
1.	309	316
2.	526	532
3.	742	749

A comparison of the maximum number of floating point operations performed by any processor for FLO52 and FLO52-Filtering as a function of the number of perrelaxation iterations,μ.

labelled 'ci'. Finally, we assume that for each element of the 'V' cycle calculation, one grid point is assigned to each processor. The processors assigned to each concurrent relaxation subproblem are indicated on the figure.[2]

The operations per iteration of each algorithm are estimated using a timing model developed for an actual hypercube implementation of FLO52 (see [1]). In Table 2, we give the operation counts corresponding to a 5 grid version of the two methods as a function of the number of prerelaxation Runge-Kutta sweeps,μ, that are performed. Notice that the filtering algorithm is more expensive, primarily due to the additional costs of splitting the residual into two components.

In addition to computing the time per iteration, we use the operation counts to determine the number of concurrent relaxation iterations that can be performed in parallel with the coarse grid correction. That is, the maximum number of concurrent relaxations that can be completed during the coarse grid correction is given by:

$$(35)\ \#\text{ concurrent iterations on } \mathcal{G}_k = \text{floor}\left[\frac{\text{time for coarse grid correction for } \mathcal{G}_k}{\text{time for one fine grid relaxation on } \mathcal{G}_k}\right].$$

In this way, we ensure that the concurrent iterations do not increase the overall time per iteration of the algorithm. In Table 3, we give the number of concurrent iterations that can be performed for each subproblem in Fig. 3 as a function of the number of prerelaxation used, μ.

8. Performance Comparisons. A series of convergence experiments were run on a serial machine to evaluate the numerical properties of both the FLO52 and the FLO52-filtering code. For the most part, the relative performance of the two algorithms was

[2] Notice that for the concurrent iteration problem on grid $\mathcal{G}_0$, each processor contains two grid points. This is because there are not enough processors available to assign one processor for each grid point.

μ \grid	$\mathcal{G}_0$	$\mathcal{G}_1$	$\mathcal{G}_2$	$\mathcal{G}_3$
1.	2	3	2	1
2.	4	6	4	2
3.	6	9	6	3

Number of concurrent iterations performed for each subproblem in Fig. 3 as a function of the number of Runge-Kutta prerelaxation sweeps, μ.

μ	FLO52	FLO52-Filtering
1.	75	42
2.	39	26
3.	26	20

Comparisons of multigrid iterations for the FLO52 and FLO52-filtering algorithms where iterations corresponds to the total (including 1, 2, 3, 4, and 5 level multigrid iterations).

fairly consistant for different grids and over a range of MACH numbers. In this paper, we present results obtained from runs on a 256 x 64 grid. For these experiments an angle of attack of .2 and a MACH number of .8 was chosen. As previously mention, a 5 level multigrid scheme is used. The computation starts on the coarsest level, $\mathcal{G}_4$, using just relaxation until the norm of the residual is reduced below 10^{-2}. This solution is interpolated to $\mathcal{G}_3$ where a multigrid procedure is used to reduce the norm of the residual to 10^{-4}. The processes is repeated for grids: $\mathcal{G}_2$, $\mathcal{G}_1$ and $\mathcal{G}_0$ reducing the residual norm to $10^{-5}, 10^{-6}$, and 10^{-9} respectively. In Table 4, we present our results for this problem using different numbers of prerelaxation sweeps (μ) within the multigrid scheme.[3] As the table illustrates, the number of iterations required for the filtering algorithm is significantly less than the number of iterations for the standard method. It should also be noted that the relative savings in using the filtering approach over the standard approach is reduced when more prerelaxation is used. This is to be expected as the primary function of the new subproblems is to reduce high frequency errors in parallel with the coarse grid correction. However, when many prerelaxation sweeps are performed (before forming the new subproblem), the high frequency error is significantly reduce before the coarse grid

[3] These results are representative of a number of different runs using different grid sizes, Mach no., and angle of attack.

correction begins.[4] Finally, we remark that it may be possible to boost the performance of the filtering method further by the use of more sophisticated operators. For example, the pre and concurrent relaxation operators use the identical Runge-Kutta coefficients that are used within the standard FLO52 scheme. These coefficients were specially chosen for the FLO52 method. Thus, improvements may be possible if a new set of coefficients is chosen for the new method. Likewise, a more carefully chosen splitting operator, Z, may also yield significant improvements.

9. Conclusion. We have presented an FAS version of the filtering method. The principle idea is to use processors that would otherwise be idle in a standard multigrid method to perform relaxation iterations. These additional iterations are performed on subproblems which are created by splitting the residual into 'smooth' and 'oscillatory' components. The 'smooth' component is used for the coarse grid correction, while the 'oscillatory' component is used for the new subproblem. By using these otherwise idle processors, a filtering iteration need not cost significantly more than standard multigrid iterations. Additionally, we remark that the modifications necessary to implement a filtering algorithm from an FAS algorithm are relatively straight-forward. This is primarily because no linearization of the operator is necessary,

We have applied the filtering approach to the FLO52 algorithm for solving the steady 2-dimensional Euler equations. Based on numerical experimentation, it has been determined that this filtering method requires fewer iterations than the standard method, and consequently, it is an attractive alternative on parallel computers.

[4] An alternative filter which distributes more of the middle frequency errors on the fine grid subproblem may yield better performance.

REFERENCES

[1] , *Performance of a Parallel Euler Equation Code On Hypercube Computers*, Tech. Rep., NASA Ames, Moffett Field, Ca, To appear 1989.

[2] , *Multi-level adaptive solutions to boundary-value problems*, Math. Comp., 31 (1977), pp. 333–390.

[3] , *Multigrid Tutorial*, SIAM, Philadelphia, 1987.

[4] , *Design and implementation of parallel multigrid algorithms*, in Proceedings of the Third Copper Mountain Conference on Multigrid Methods, S. McCormick, ed., Marcel Dekker, NY, 1987, pp. 101–115.

[5] , *Constructive interference in parallel algorithms*, SIAM Journal on Numerical Analysis, 25 (1987), pp. 376–398.

[6] , *Parallel superconvergent multigrid*, in Proceedings of the Third Copper Mountain Conference on Multigrid Methods, S. McCormick, ed., Marcel Dekker, NY, 1987, pp. 195–210.

[7] , *A new approach to robust multi-grid methods*, in First International Conference on Industrial and Applied Mathematics, Paris, 1987.

[8] , *Solution of the euler equations for two dimensional transonic flow by a multigrid method*, Appl. Math. and Comp., 13 (1983), pp. 327–335.

[9] , *Multigrid methods for partial differential equations*, in Studies in Numerical Analysis, G. Golub, ed., 1984, pp. 270–318.

[10] , *Multigrid Algorithms on Parallel Processing Systems*, PhD thesis, Stanford University, 1989.