JOURNAL OF COMPUTATIONAL PHYSICS **125,** 338–353 (1996)
ARTICLE NO. 0098

Scalability and Performance of Data-Parallel Pressure-Based Multigrid Methods for Viscous Flows

EDWIN L. BLOSCH AND WEI SHYY

Department of Aerospace Engineering, Mechanics and Engineering Science, University of Florida, Gainesville, Florida 32611

Received May 17, 1995; revised September 25, 1995

A full-approximation storage multigrid method for solving the steady-state 2-*d* incompressible Navier–Stokes equations on staggered grids has been implemented in Fortran on the CM-5,using the array aliasing feature in CM-Fortran to avoid declaring fine-grid-sized arrays on all levels while still allowing a variable number of grid levels. Thus, the storage cost scales with the number of unknowns,allowing us to consider significantly larger problems than would otherwise be possible. Timings over a range of problem sizes and numbers of processors, up to 4096×4096 on 512 nodes, show that the smoothing procedure, a pressure-correction technique, is scalable and that the restriction and prolongation steps are nearly so. The performance obtained for the multigrid method is 333 Mflops out of the theoretical peak 4 Gflops on a 32-node CM-5. In comparison, a single-grid computation obtained 420 Mflops. The decrease is due to the inefficiency of the smoothing iterations on the coarse grid levels. W cycles cost much more and are much less efficient than V cycles, due to the increased contribution from the coarse grids. The convergence rate characteristics of the pressure-correction multigrid method are investigated in a $\mathrm{Re} = 5000$ lid-driven cavity flow and a $\mathrm{Re} = 300$ symmetric backward-facing step flow, using either a defect-correction scheme or a second-order upwind scheme. A heuristic technique relating the convergence tolerances for the coarse grids to the truncation error of the discretization has been found effective and robust. With second-order upwinding on all grid levels, a 5-level 320×80 step flow solution was obtained in 20 V cycles, which corresponds to a smoothing rate of 0.7, and required 25 s on a 32-node CM-5. Overall, the convergence rates obtained in the present work are comparable to the most competitive findings reported in the literature. © 1996 Academic Press, Inc.

1. INTRODUCTION

The convergence rate of iterative methods for solving the steady-state incompressible Navier–Stokes equations can be greatly improved using multigrid acceleration techniques, which were first described in this context by Brandt [6]. Multigrid methods work by damping both oscillatory and smooth solution error components quickly, and hence are especially advantageous for flow problems with disparate physical scales—rapidly varying errors can be damped quickly on fine grids, while smooth errors can be damped quickly on coarser grids. The main advantage of using such techniques is that much greater grid resolution can be used in all or part of the domain without sacrificing the convergence rate of a less-resolved problem.

Parallel computing is advantageous because of the large problem sizes which can be accommodated. The computational speeds (e.g., Mflops) which can be obtained, however, are strongly algorithm-dependent and, for a particular algorithm, strongly problem-size dependent. The effective computation rate depends on the relative amounts and speeds of computation and interprocessor communication. Raw communication speeds are typically orders of magnitude slower than floating-point operations. Thus more often than not the communication steps in the algorithm, and the network performance for these steps, strongly influence the parallel run time.

Because of the aforementioned advantages of multigrid methods and parallel computing, there has been much interest in developing and testing parallel multigrid programs, as summarized in Refs. [31, 35, 58]. A key issue, one which appears in more than one context in parallel computing, is scalability. A scalable parallel *architecture* is one whose interprocessor communication network provides a fixed bandwidth for a particular communication operation, e.g., all-to-all broadcast, as the number of processors grows. Scalable parallel *algorithms* are, in an absolute sense, those whose computational and communication complexity both depend only on the problem size per processor. All the processors must be involved for all the steps. In a relative sense, scalable algorithms are those which obtain linear speedup in the scaled-size experiment [23, 28]. However, the introduction of the idea of speedup to assess scalability is somewhat painful, since it depends on how one defines speedup [22]. Scalable *implementation*, yet another context, refers to the memory requirements of the parallel program, and how these increase with the number of processors.

Real problem-solving programs consist of many constituent algorithms, any or all of which may have to be iterated many times as part of the solution procedure. In the context of solving the steady-state Navier–Stokes equations, the number of iterations is unknown beforehand and is problem-dependent. Consequently, numerical experiments and timings on test problems, for a range of problem sizes and numbers of processors, are essential for identifying and clarifying the relevant issues. In this research a multigrid algorithm, using a sequential pressure-based solution procedure to solve the governing equations on each grid level, has been implemented on a CM-5 using single-instruction stream/multiple-data stream data-parallelism. The algorithm's performance is discussed from two perspectives, first from the standpoint of computational efficiency and scalability, and second from the standpoint of the convergence rate characteristics. We have studied the cost per cycle for fixed V and W multigrid cycles, using timings for a range of problem sizes and numbers of processors, up to 4096 × 4096 on 512 nodes. V cycles are better in terms of cost on the CM-5, and good convergence rates can be attained when the initial fine-grid guess is generated by a nested-iteration ("full-multigrid") procedure. We demonstrate and discuss this point in the context of a Re = 300 symmetric backward-facing step flow and a Re = 5000 lid-driven cavity flow. Also, comparisons are made between the defect-correction technique and second-order upwinding. Both methods give a formally second-order accurate discretization on the finest grid level and are stable for the high cell Peclet number problems that arise on the coarse grids. However, they impact the convergence rate and stability of multigrid iterations in different and problem-dependent manners.

2. BACKGROUND

2.1. *Description of Multigrid Scheme and Smoothing Procedure*

The components of a multigrid method are the smoothing procedure by which the equations on each grid level are solved, and the restriction and prolongation procedures by which the equation residuals and correction quantities are transferred between grid levels. The multigrid scheme specifies how coarse-grid problems are generated from the fine-grid problem and in what order the multiple grid levels are visited, i.e., the cycle type. Briefly, one multigrid V iteration is a recursive algorithm which has the following steps: pre-smoothing iterations, restriction to a coarse grid, solving the coarse grid problem for coarse-grid corrections, prolongation of corrections to the fine grid, and post-smoothing iteraitons. The coarse-grid problems are solved by recursion, which leads to the V shape of the cycle, as shown in Fig. 1. In the figure the number of pre-smoothing and post-smoothing iterations are shown inside the circles; it is a fixed V(3,2) cycle.

For linear equations such a procedure is typically very efficient, depending on how fine-grid solution errors are damped through the process of smoothing and coarse-grid correction. Briggs [9] gives both spectral and algebraic explanations which show precisely how fine-grid solution errors are eliminated in a few V cycles for the Poisson equation using point-Jacobi smoothing iterations and bilinear restriction and prolongation procedures, and there are several good reviews of multilevel algorithms in a variety of other contexts as well [15, 16, 26, 10].

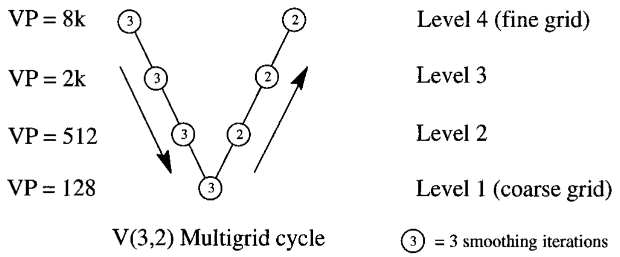


FIG. 1. Schematic of a V(3,2) multigrid cycle.

W cycles are generated when the coarse-grid corrections are updated twice instead of once before being interpolated to the fine grid, as shown in Fig. 2. The convergence acceleration of multigrid comes from the coarse-grid corrections [16] and thus W cycles usually converge faster. However, larger corrections require more post-smoothing to avoid stability problems and so the additional cost of W cycles is an issue, an important one for SIMD-style parallel computation.

The cost of one V cycle using n_{level} levels and $(n_{\text{pre}}, n_{\text{post}})$ pre- and post-smoothing iterations can be modelled as

$$\frac{\text{Time}(s)}{V\,\text{cycle}} = \sum_{k=1}^{n_{\text{level}}} s_k(n_{\text{pre}} + n_{\text{post}}) + \sum_{k=2}^{n_{\text{level}}} (r_k + p_k), \qquad (1)$$

where s_k, r_k, and p_k are the times for one pressure-correction iteration on level k, for restriction from level k to level $k - 1$, and for prolongation to level k from level $k - 1$. For W cycles, the run time can be modelled as

$$\frac{\text{Time}(s)}{W\,\text{cycle}} = \sum_{k=1}^{n_{\text{level}}} s_k(n_{\text{pre}} + n_{\text{post}})2^{(n_{\text{level}}-k)} + \sum_{k=2}^{n_{\text{level}}} (r_k + p_k)2^{(n_{\text{level}}-k)}. \qquad (2)$$

Thus the number of smoothing iterations on the coarsest grid ($k = 1$) increases geometrically with the number of levels for a W cycle. On a serial computer it is generally reasonable to neglect the restriction and prolongation cost in comparison with smoothing, and to assume that the smoothing cost is proportional to the number of unknowns; i.e., the cost on level $k - 1$ is $\frac{1}{4}$ of the cost for level k in 2-d. Furthermore, this model allows one to measure the multigrid cycle cost in terms of work units, equivalent to

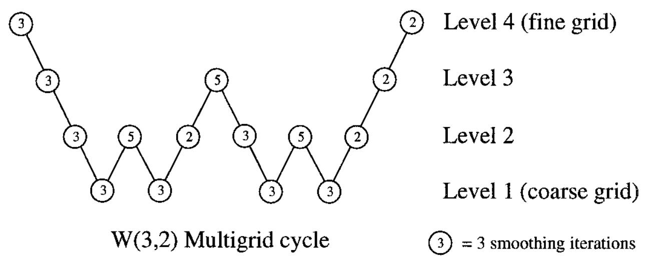


FIG. 2. Schematic of a W(3,2) multigrid cycle.

fine-grid iterations. The cost on parallel computers must be measured because the assumption of equal smoothing efficiency regardless of problem size is no longer valid.

The full-approximation storage (FAS) scheme, described by Brandt [6], is used to form the coarse-grid equations taking account of the nonlinearity of the fine-grid problem. In this scheme, one obtains an approximation to the full fine-grid solution, on the coarse grids, by adding appropriately averaged source terms to the coarse-grid equations. The coarse-grid problem is treated the same as the fine-grid one, by discretizing the original equations on the coarse grid. Thus the same solution technique for the Navier–Stokes equations is used on all grid levels, with the additional numerically derived source terms treated like physical source terms.

If the governing equations are fully coupled and solved by iteration, then the solution of the linearized equations can employ the "correction" scheme, wherein coarse-grid equations for the errors only, not the full solutions, are derived directly from the discretized fine-grid equations, usually by the Galerkin approximation [62]. Correction schemes may also be used for the pressure equation [5, 25, 42, 43] in a sequential solution procedure. In the latter context, however, the convergence rate deterioration with increasing problem size of the outer iterations persists because the velocity–pressure coupling is not addressed. In unsteady flow algorithms multigrid correction schemes are also used, e.g., in projection methods [12, 37] to solve the pressure-Poisson equation. The important distinction is again that the steady-state discretized equations or equivalently the fully implicit time-dependent equations do not boil down to solving a linear system of equations in the sequential pressure-based algorithms. We solve the full set of equations on each grid level and couple the grid levels using FAS, whereas in time-dependent methods multigrid is typically applied to accelerate the solution of the pressure Poisson equation. Thus, for the latter, multigrid techniques make the cost per time-step scalable but the time-step size is still restricted by the problem size through the viscous stability constraint.

With FAS the discrete problem on each grid level is obtained by discretizing the differential equations. However, it is not always possible to use the same discretization on each grid level. For example, central-differencing the convection terms is likely to be unstable on coarse grids in most problems, depending on the boundary conditions [45]. The multigrid iterations will diverge if the smoothing iterations diverge on the coarse grid(s). However, it is also difficult to use central-differencing on the fine grid for accuracy and first-order upwinding on the coarse grids, unless special restriction and prolongation procedures are employed, or unless the convection effect is not dominant. We have studied two "stabilization strategies" that deal with this problem, namely the use of defect corrections as in Refs. [50, 55, 57, 3, 2] and the use of an unconditionally stable second-order convection scheme on each grid level, i.e., second-order upwinding. These techniques and the restriction and prolongation procedures used are presented in conjunction with the results.

The smoother used is the semi-implicit method for pressure-linked equations (SIMPLE [40]), which along with SIMPLEC [14] and SIMPLER [39] is described as a sequential pressure-based method for the steady-state incompressible Navier–Stokes equations. The term smoother is used in the multigrid literature to emphasize the role of the solution method as one of eliminating only those errors which are oscillatory on the scale of the grid level under consideration. However, in the present context, where, due to the nonlinearity of the governing equations, errors propagate through the multigrid cycle in unpredictable manners, there is much discussion about which numerical methods for the steady-state incompressible Navier–Stokes equations perform best in the FAS multigrid setting [61]. In the sequential pressure-based methods, the velocity components are (separately) updated by applying several iterations of a point- or line-iterative method to the implicitly discretized momentum equations, with pressure lagged. An approximate pressure-Poisson equation is derived from the discrete momentum and continuity equations to correct both the pressure and velocity fields. An approximate solution is obtained by taking a few iterations. Thus after one iteration neither mass nor momentum is conserved but it was not necessary to solve the pressure-Poisson equation to strict tolerances. Due to the approximation made in the pressure–velocity correction relationship, these outer iterations are repeated until convergence. Thus, the computational nature of the method is best described as double-iterative. The convergence rate depends on the inner iterations, the underrelaxation factors, the Reynolds number, the strength of pressure–velocity coupling in the flow problem, and the grid distribution.

Recently we have implemented the single-grid method on single-instruction stream/multiple-data stream computers such as the MasPar MP-1 and Connection Machine CM-5 [4]. Because every iteration is a synchronization point and there need to be frequent global sums to compute equation residuals and monitor convergence, the SIMD model is a natural match. With regard to performance, though, note that the essential character of the algorithm, namely its division into communication and computation operations and the mapping of data to processors, would be the same on any distributed memory machine. The key is the data-parallel approach to parallelizing the algorithm, not the SIMD vs MIMD distinction [24]. For large problem sizes, the single-grid computational speeds that were measured (420 MFlops) were consistent with others' results [17, 27], and achieved parallel efficiencies approaching 0.8. We used point-Jacobi inner iterations, which require only local updating of variables. Consequently, the run time per iteration was scalable in the absolute sense. Also, to make the cost per iteration scalable, it was necessary to compute equation coefficients using a uniform formulation to do the boundary control volumes simultaneously with the interior control volumes.

In many cases the single-grid SIMPLE method with point-Jacobi inner iterations requires small relaxation factors and consequently converges slowly. However, with multigrid, good convergence rates are attainable, as our results show. Furthermore, the multigrid procedure can add robustness in comparison to the single-grid SIMPLE method [46]. Recent work on nonlinear monotone convection schemes [51] and composite-overlapping grid algorithms [59] has added significant flexibility to the solution methodology. Sequential pressure-based methods work for both incompressible and compressible flow regimes, and have been applied to a variety of aerodynamic and phase change/heat transfer problems. See Ref. [45] for background and an extensive list of references.

Vanka [57] has developed a locally coupled explicit method for the steady-state incompressible Navier–Stokes equations. As a single-grid solver it is slowly converging, but in the multigrid context it is evidently a good smoother. Linden *et al.* [32] reviewed multigrid methods for the steady-state incompressible Navier–Stokes equations and stated a preference for Vanka's method (called block-Gauss–Seidel, BGS) over sequential methods, but no direct comparisons were made. Sockol [50] has compared the performance of BGS, two line-updating variations on BGS, and the SIMPLE method with successive line-under-relaxation for the inner iterations on three model flow problems with different characteristics, varying grid aspect ratios, and a range of Reynolds numbers. Each method was best, in terms of work units, in some range of the parameter space. Brandt and Yavneh [8] have studied the error smoothing properties of a multigrid method which uses a line-iterative sequential smoother for the incompressible Navier–Stokes equations. Good convergence rates were observed for "entering-type" flow problems in which the flow has a dominant direction and is aligned with grid lines. Evidently, line-relaxation has the effect of providing non-isotropic error smoothing properties to match the physics of the problem. These researches indicate that sequential pressure-based methods are viable multigrid smoothers, and that the competitiveness with Vanka's smoother is a problem-dependent consideration. Other experiences with steady-state incompressible flows via multigrid methods include Refs. [34, 60].

For parallelization we have implemented Vanka's method with a red–black updating scheme, so we call the method "block-red–black," BRB, by analogy. Although we have not made convergence rate comparisons because of the problem-dependency of the results, we have compared the cost per iteration. Timings show that the cost per iteration is virtually the same as BRB when 3, 3, and 9 inner point-Jacobi iterations are used for SIMPLE [3]. On serial computers, it is cost-effective to use line-relaxation in the inner iterations for the SIMPLE method. On the CM-5, however, it is relatively less attractive in comparison with point-wise relaxation—one line-Jacobi iteration takes roughly four times as long as a point-Jacobi iteration. Line-variants on Vanka's scheme are possible but not competitive in CM-Fortran because prohibitively expensive interprocessor communications are necessary.

2.2. *Parallel Multigrid Issues*

Parallel multigrid methods in CFD [1, 20, 21, 31, 33] have received much attention recently motivated by their almost ideal scalability ($O(\log^2 N)$ for problem size N on $n_p = N$ processors, in theory) on Poisson equations. On parallel computers, however, the computational complexity is not an accurate reflection of the run time, because all computations are not carried out with equal efficiency and because the contribution from communication is ignored. One of the biggest practical concerns is the nature of the workload generated by the smoothing iterations on coarse grid levels, which is characterized by very low ratios of computation to communication work. The degree to which this affects the overall efficiency of a given multigrid cycle depends on the particulars of the parallel machine and the algorithm, although the effect is the same for any data-parallel computation mapped in a blockwise fashion.

Several ideas are being explored to improve the efficiency with which the coarse grid level work is carried out. One idea is to use multiple coarse grids to increase the efficiency of the computations and communications (see Refs. [38, 20, 49] and the references therein for recent research along this line). In absolute terms, more computation and communication are done per cycle, but more efficiently if the multiple coarse grids are treated in parallel. The obvious problem is making the extra work pay off in terms of convergence rate. Another idea is to alter the grid-schedule to only visit the coarsest grid levels every couple of cycles. This approach can lead to nearly scalable implementations [21, 30] but may sacrifice the convergence rate. "Agglomeration" is an efficiency-increasing technique used in MIMD multigrid programs which refers to the technique of duplicating the coarse grid problem in each processor so that computation proceeds independently (and redundantly). Such an approach can also be scalable [33].

The degree to which the aforementioned ideas improve the fundamental efficiency/convergence rate tradeoff is unknown. There have also been efforts to devise novel multilevel algorithms with more parallelism for SIMD computation [18, 19, 20]. These efforts and others have recently been reviewed [11, 35, 58]. Most of these algorithms are still untested in the context of the incompressible Navier–Stokes equations.

Another concern is the multigrid storage problem. Dendy *et al.* [13] have recently described a multigrid method on the CM-2. However, to accommodate the data-parallel programming model they had to dimension their array data on every grid level to the dimension extents of the fine-grid arrays. This approach is very wasteful of storage, but is difficult to avoid with distributed memory parallel computers. Consequently the size of problems which can be solved is greatly reduced. Recently an improved release of the CM-Fortran compiler has enabled the storage problem to be circumvented with some programming diligence. The storage issue and the implementational trick we have used are described in the Appendix. Our implementation has the same storage requirements as serial multigrid algorithms. The High-Performance Fortran (HPF) compilers currently becoming available will allow the same approach to be employed for multigrid algorithms on many distributed-memory machines.

3. RESULTS

The CM-5 used in the present work is a collection of SPARC processing nodes, each with four attached vector floating-point units. Each vector unit is a computer and a memory manager for 32 MBytes. Thus, a 32-node CM-5 actually has 128 independent processing elements. The SPARC nodes are connected by a special "control network" which is automatically exploited to provide synchronization of the nodes' execution and to compute global reduction-type operations when the CM-Fortran language is used. A separate "data network" interconnects the processors by a fat-tree [52].

CM-Fortran, which exploits array-based data-parallelism, is sufficient for the present application. Using CM-Fortran, synchronization and interprocessor communication are accomplished by compiler-generated calls to a run-time communication library. It is also possible to program the CM-5 using explicit message-passing, if needed for example for load-balancing considerations, but that

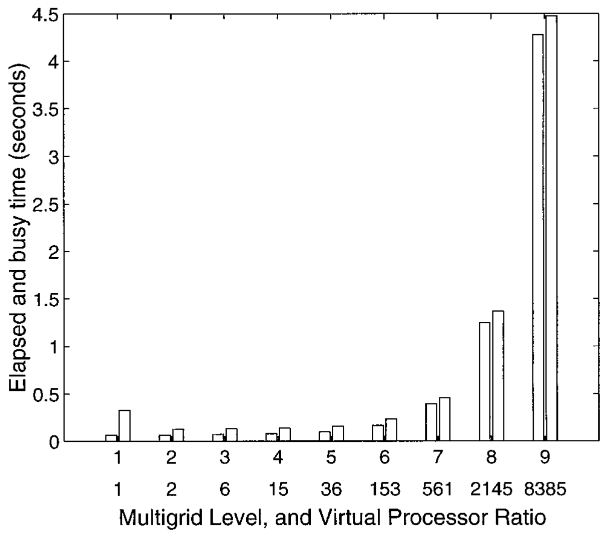


FIG. 3. Smoothing cost, in terms of elapsed and busy time on a 32-node CM-5, as a function of the multigrid level for a case with a 1024 × 1024 fine grid. The elapsed time is the one on the right (always greater than the busy time). The times correspond to five SIMPLE iterations.

option was not needed here. We are further motivated to choose CM-Fortran over serial Fortran/C combined with explicit message-passing to take advantage of recent developments in portable parallel compilers. Recently, Applied Parallel Research released an HPF compiler which can compile the NAS parallel benchmarks on the Cray T3D, IBM SP-2, and Intel Paragon parallel computers [44]. CM-Fortran is essentially a subset of HPF, and thus we are optimistic that our code will be highly portable. Although SIMD *architectures* have lost ground to MIMD and shared memory, the data-parallel computational model which CM-Fortran exploits is ubiquitous and probably will increase in popularity as portable data-parallel compilers mature.

3.1. *Efficiency and Scalability on the CM-5*

Inefficiency of the Coarse-Grid Smoothing and Implications. Figure 3 displays the relative cost of the smoothing iterations for the coarse grid levels, for a representative multigrid calculation on 32 nodes. The finest grid level, level 9, corresponds to a 1024 × 1024 grid, and the coarsest grid level corresponds to 4 × 4. The times given are for five SIMPLE iterations; i.e., they correspond to one V(3,2) cycle, using 3, 3, and 9 inner point-Jacobi iterations on the u, v, and pressure-correction equations, respectively. The smoothing cost would be translated up or down depending on the number of pre-and post-smoothing iterations per cycle.

From the figure, one can see that for the CM-5 the coarse-grid levels' smoothing cost, while still small in comparison with the cost of fine-grid pressure-correction iterations, is no longer negligible. In fact, there is basically no decrease in the smoothing times beneath level 5, which corresponds to a 128×128 grid. In Fig. 3, the bar on the left is the CM-5 busy time and the bar on the right is the corresponding elapsed time. Busy time is the time spent doing parallel computation and interprocessor communication operations. The elapsed time is the busy time plus the contribution from front-end-to-processor communication, i.e., the passing of code blocks, which is effectively a constant overhead. Beneath level 5, this overhead dominates, and consequently we do not see the coarse-grid smoothing cost (elapsed time) go to zero as VP goes to zero, as is the case for serial computation.

Nor do the busy times scale linearly, but this is related to the efficiency of interprocessor communication and vectorized computation. For nearest-neighbor interprocessor communication of arrays that are mapped onto the processor mesh by contiguous blocks, there are effectively three different communication speeds, because (1) some of the data stay on the same vector unit; (2) some data must move between vector units but stay on the same SPARC node; and (3) some of the data must move between SPARC nodes. Thus the communication performance and, due to the vector units, the computation performance, are strongly dependent on the problem size. In previous work [4] we have found that on the CM-5 the peak efficiency of smoothing iterations is not obtained until the problem size is greater than about 2K unknowns (per processor). For large problem sizes, the best performance is achieved because the vector units are full and surface-area/volume effects make effectively all communication of the fastest type.

In the present application, only computation is useful work. Therefore, coarse-grid smoothing iterations, which contain proportionately less computation but the same overheads as fine-grid iterations, are very inefficient on the CM-5. In a W cycle, a grid level k is visited $2^{n_{\text{level}}-k}$ times (see Eq. (2)), as compared to twice for a V cycle. Figure 4 plots the elapsed and busy times for 7 level V (3,2) and W(3,2) cycles, as measured on the 32-node CM-5, against the virtual processor ratio of the finest grid level. The number of levels is kept fixed as the finest grid dimensions increase. Evidently, the geometric increase in frequency of visitation has a significant effect.

The run time per cycle is approximately three times that of the V cycle. For large VP, efficiency is approximately the ratio of busy to elapsed time, and from the figure one can infer that the W cycle efficiency is much less than the V cycle efficiency. Furthermore, since the frequency of visitation of coarse grids scales geometrically with the number of grid levels, the efficiency of W cycles decreases as the problem size increases, if additional levels are added. In contrast, V cycles are nearly scalable, as discussed in the next section. To summarize, it is no longer clear on the CM-5, in comparison with serial computations, that the convergence rate benefit of W cycles can be gained at reasonable expense. Repeating the experiment with 5 or 3 levels shows that the coarse grids impact the efficiency less, but of course the convergence rate using a truncated number of levels would be poorer. Generally the tradeoff between V and W cycles, and the choice of the number of levels, is problem-dependent, but the tradeoff is shifted in favor of V cycles on the CM-5 in comparison to the situation for serial computations. For the flow problems considered here, satisfactory convergence rates have been obtained using V cycles with a special full-multigrid strategy to be described shortly.

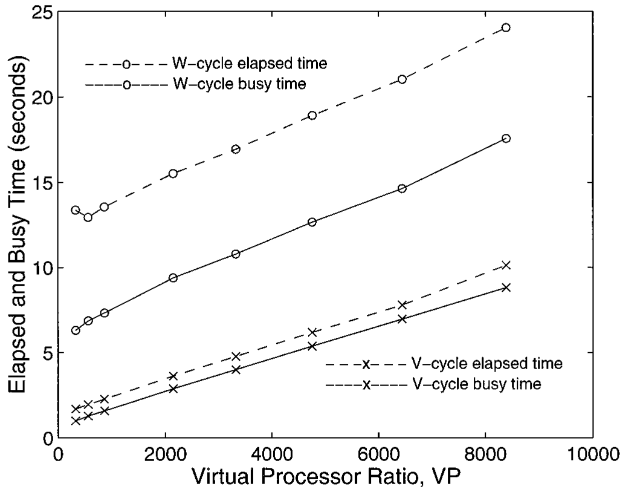


FIG. 4. Elapsed and busy time per cycle on a 32-node CM-5, as a function of the problem size, given in terms of VP. The number of levels is fixed at 7 as the dimensions of the finest grid are increased.

Scalability of Restriction and Prolongation Operations. As for serial computation, the contribution to the cost per cycle from the restriction and prolongation operations is small, as can be seen from Fig. 5. The restriction cost is not shown for clarity but is slightly less than prolongation and shows the same trend. The ratio of the times for restriction, prolongation, and smoothing tends toward 1 : 2 : 13 on the 32-node CM-5, as the problem size increases. Obviously, the relative contribution from restriction and prolongation will depend on the amount of smoothing.

The cost of the intergrid transfer steps scales linearly with problem size, as was the case for smoothing (for VP > 32), which reflects the fact that restriction/prolongation are just local averaging/interpolation operations applied to every residual. However, this cost does depend on the number of processors. The CM-5 fat-tree network is involved in this step because restriction and prolongation inevitably require parallel-array expressions involving two arrays which are declared to have different dimensions,

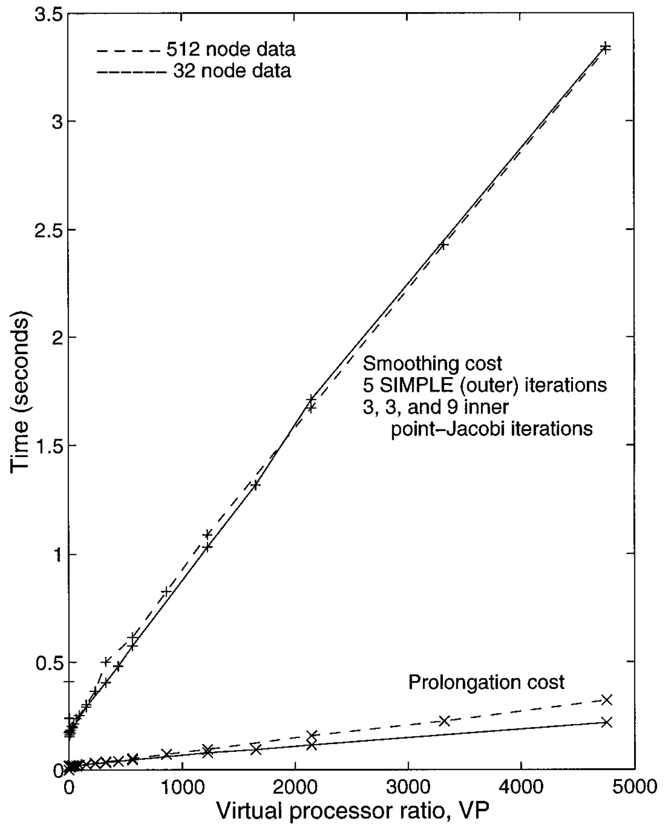


FIG. 5. Smoothing and prolongation times per V-cycle, as a function of the problem size, for 32 and 512-node CM-5 computers (128 and 2048 processing elements, respectively). The times are elapsed times, for V(3,2) cycles: 5 smoothing iterations, 1 restriction, and 1 prolongation, at each grid level. The restriction cost is not shown for clarity. The trend is the same as for prolongation and the time is roughly the same.

i.e., one which is defined on the coarse grid and one which is defined on the fine grid. The mapping of arrays to processors is done at run time, as it must be since the number of processors is unknown a priori. Thus, the compiler has no knowledge that would enable it to recognize that the restriction or prolongation expression could probably be accomplished with a regular communication pattern. It must assume that the communication pattern is general, and let the run-time communication library do the routing. In the example above, we find that the prolongation to the grid level with VP = 4K, which is roughly 360 × 360 on 32 nodes and 1440 × 1440 on 512 nodes, takes 0.4 s on 32 nodes but 0.5 s on 512 nodes.

Evidently, the performance of the global data network is not perfectly scalable for the prolongation. According to Ref. [52], the global communication network of a 32-node CM-5 is a fat tree of height 3. The fat tree is similar to a binary tree except the bandwidth stays constant upwards from height 2 at 160 MBytes/s. In practice, one can expect approximately 480 MBytes/s for regular grid communication patterns (i.e., between nearest-neighbor SPARC nodes) and 128 MBytes/s for random (global) communications. Thus, it is really only true to say that the performance of the network is scalable beyond height 2. The restriction and prolongation routines evidently generate communications which travel farther up the tree on 512 nodes than on 32 nodes. The net effect is that a prolongation from a grid level with VP = 1K to a grid level with VP = 4K takes longer on 512 nodes, although not by much.

From the relatively small contribution made by restriction and prolongation it follows that multigrid cycles are nearly scalable, as shown in Fig. 6. This figure plots the isoefficiency metric of scalability as described in [29, 28]. Curves of constant parallel efficiency were drawn for a range of problem sizes and number of processors using linear least-squares curve fits to the timing data. The isoefficiency curves are almost linear or, in other words, the 7-level multigrid algorithm analyzed on a per-cycle basis, is almost scalable. Each of the isoefficiency curves can be accommodated by an expression of the form

$$N - N_0 = \text{constant } (n_p - 32)^\alpha, \tag{3}$$

with $\alpha \simeq 1.1$. The symbol N_0 is the initial problem size needed to obtain a particular E on 32 processors.

Along perfectly straight isoefficiency curves, "scaled-speedup" [23, 22] is achieved. In other words, if the parallel run time T_p at some initial problem size and number of processors is acceptable, then it can be maintained as the problem size and the number of processors are increased in proportion. We also note that although we have used point-Jacobi inner iterations, we expect similar scalability

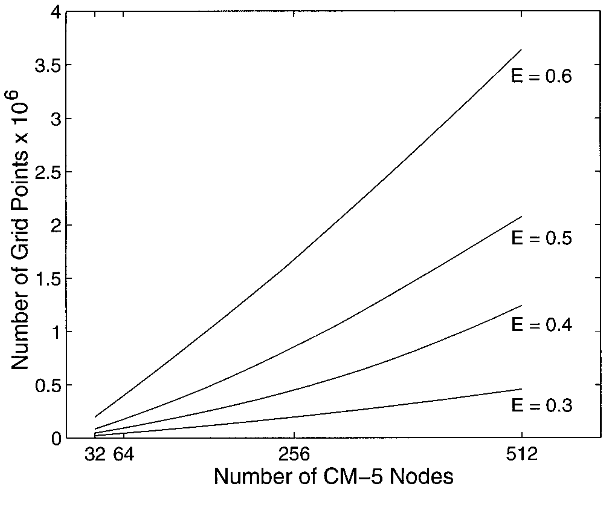


FIG. 6. Isoefficiency curves for the 7-level pressure-correction multigrid method, based on timings of a fixed number of V(3,2) cycles, using point-Jacobi inner iterations. The isoefficiency curves have the general form $N = \alpha n_p^\beta +$ constant, where $\beta \simeq 1.1$ for the efficiencies shown.

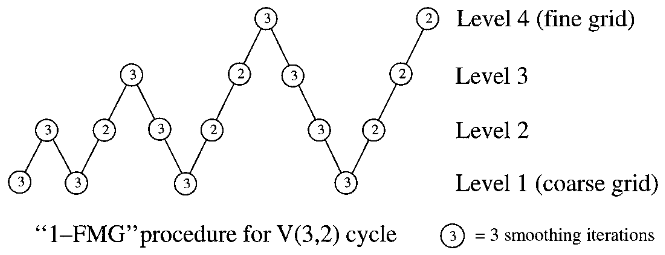

Level 4 (fine grid)
Level 3
Level 2
Level 1 (coarse grid)
"1–FMG" procedure for V(3,2) cycle
3 = 3 smoothing iterations

FIG. 7. Schematic of an FMG V(3,2) multigrid cycle.

(although longer actual run times given a particular problem size) with line-Jacobi inner iterations, based on previous comparisons between the two in the context of single-grid computations [4]. Since the line-iterative method is $O(N\log_2 N)$, T_p would increase slightly along the isoefficiency curves, but the effect due to the logarithmic factor is generally quite small due to the early cut-off tolerance in the cyclic reduction kernel used in the line solver, which reduces the amount of long-distance communication.

3.2. *Multigrid Convergence Rate Characteristics*

The total solution cost is the number of iterations multiplied by the cost per iteration, and viable paths to massively parallel computing requires scalability in both dimensions. For linear model equations, multigrid methods can obtain convergence rates which show perfect problem-size independence. However, for fluid flow problems, ideal performance cannot be achieved in general and there are many unresolved issues, as outlined in the background section. Thus our focus has been on developing understanding of important factors affecting the convergence rate and stability.

Truncation Error Control of FMG Cycling and Implications. The initial guess for the fine-grid solution impacts the convergence rate, and so a nested iteration approach for generating the starting fine-grid solution, also called a "full-multigrid" (FMG) strategy, is generally desirable. The convergence rate of FMG-V cycles is nearly scalable, for linear problems, given suitable restriction, smoothing, and prolongation procedures. In the present context the following technique is used. Starting with a zero initial guess on the coarsest grid, a few pressure-correction iterations are done to obtain an approximate solution. This solution is prolongated to the next grid level and 2-level multigrid V cycles are initiated, continuing until some convergence criterion is met. The solution is prolongated to the next finer grid and multigrid cycling (3-level this time) resumes again, and so on until the finest grid level is reached. The converged solution on level $k_{\max} - 1$, where k denotes the grid level, interpolated to the fine grid, is a better starting point than an arbitrary or zero fine-grid solution. Fig. 7 shows a "1-FMG" V(3,2) cycle for a 4-level computation. In practice, V cycles are continued on each level until the prescribed coarse-grid convergence tolerances are met.

Thus the coarse-grid tolerances figure into both the cost and the effectiveness of the FMG procedure. We have observed that both the convergence rate and the stability of multigrid iterations can be affected by the initial fine-grid guess, depending on the stabilization strategy and flow problem. With regard to cost, the FMG iterations are of more concern for parallel computation than for serial computation, due to the relative inefficiency with which coarse grids are smoothed. Usually they are still cost-effective [56], but the overall efficiency can be quite low unless the fewest possible number of coarse-grid cycles are taken.

Based on discussions in Ref. [55, 41, 7] we have developed an estimate of the solution truncation error, and it is this quantity which we monitor during the FMG cycling to assess convergence. Ultimately the FAS scheme tries to obtain the same solution at corresponding locations on every grid level by adding source terms related to the difference in truncation errors between adjacent levels in the grid hierarchy. Thus the solutions on all the coarse grids in the FMG procedure should be made close to the differential solution, i.e.,

$$\|A^h u - A^h v^h\| \le \varepsilon, \tag{4}$$

where A^h is the discrete equation operator on some coarse grid whose grid spacing is denoted by h, u is the exact differential solution, v^h is an approximate solution to the discrete problem whose exact solution is u^h, and ε is a small number. The development for the coupled system of momentum and continuity equations is given in Ref. [3], but here we discuss a single equation for brevity.

Since u is unknown, though, one can only assess the equation residual $r^h = \|A^h u^h - A^h v^h\|$. To relate the residual to the level of truncation, use the triangle inequality and the definition of truncation error,

$$\|A^h u^h - A^h v^h\| \le \|A^h u^h - A^h u\| + \|A^h u - A^h v^h\| \le \|\tau^h\| + \varepsilon \tag{5}$$

where τ^h denotes the solution truncation error on the coarse grid. Thus for small ε

$$\|r^h\| \le \|\tau^h\| \tag{6}$$

should be the convergence criterion. The truncation error is in turn related to the discretization of two adjacent grid levels, assuming a second-order scheme, as

$$\tau^h = \frac{[A^{2h}u - S^{2h}] - [A^h u - S^h]}{3}, \tag{7}$$

where S^{2h} just indicates the source term on the grid with spacing $2h$; by the problem definition, $S^{2h} = A^{2h}u^{2h}$, and likewise for S^h. Equation (7) is evaluated on the grid whose spacing is $2h$ to give the truncation error estimate for the grid whose spacing is h. Substituting the most current approximations for u, v^h and v^{2h}, Eq. (7) becomes

$$\tau^h \simeq \frac{[A^{2h}(v^{2h}) - S^{2h}] - [A^h v^h - S^h]}{3}. \tag{8}$$

The second term in brackets is just the residual r^h, which as discussed below is approximated on the grid with spacing $2h$ by summing over the appropriate control volumes for the finite-volume methodology. Thus it can be observed that the truncation error is just the numerically derived part of the source term in the discretized equation on the grid whose spacing is $2h$,

$$\tau^h \simeq \frac{S^{2h}_{\text{numerical}}}{3}. \tag{9}$$

Thus Eq. (6), the convergence criterion, is

$$\|r^h\| \le \left\|\frac{S^{2h}_{\text{numerical}}}{3}\right\|. \tag{10}$$

Two levels are always available to make this estimate for the coarse grid cycling in the FMG procedure. The L_1 norm is used, divided by the appropriate number of control volumes; for a gridvector v on an $N \times N$ mesh,

$$\|v\| = \sum_{\text{all } i,j} \frac{|v_{i,j}|}{N^2}. \tag{11}$$

Equation (10) is evaluated on the fly and without additional cost since the source term $S^{2h}_{\text{numerical}}$ is already evaluated as part of the coefficient computations for the previous multigrid cycle's post-smoothing iterations on the grid whose spacing is $2h$.

Numerical Experiments. Two flow problems with different physical characteristics have been considered, a lid-driven cavity flow at Reynolds number 5000 and a symmetric backward-facing step flow at Reynolds number 300. Streamlines, velocity, vorticity, and pressure contours are shown in Figs. 8 and 9. In the lid-driven cavity flow, convection and cross-stream diffusion balance each other in most of the domain and the pressure gradient is insignificant except in the corners. The flow is not generally aligned with the grid lines. In the symmetric backward-facing step flow, the pressure gradient is in balance with viscous diffusion, and except in the (weak) recirculation region, is strongly aligned with the grid lines. Thus, the two model problems pose different challenges for our smoothing procedure, and at the same time are representative of broader classes of flow problems.

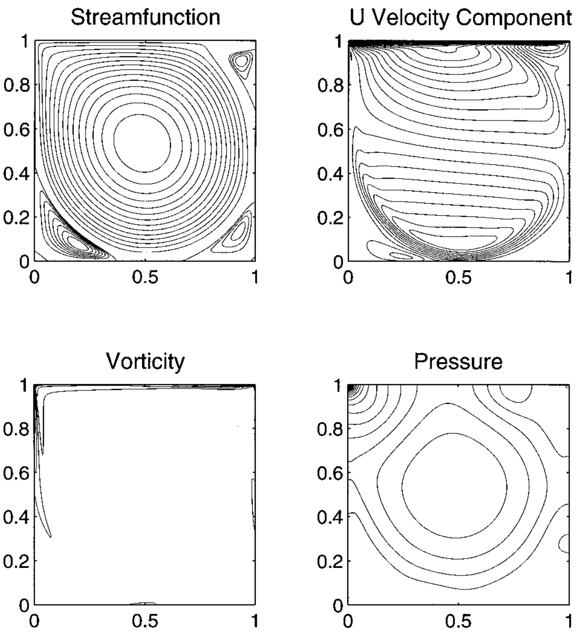


FIG. 8. Streamfunction, vorticity, and pressure contours for Re = 5000 lid-driven cavity flow, using the 2nd-order upwind convection scheme. The streamfunction contours are evenly spaced within the recirculation bubbles and in the interior of the flows, but this spacing is not the same. The actual velocities within the recirculation regions are relatively weak compared to the core flows.

Two stabilization strategies have been considered. One is the defect-correction approach as described in Ref. [50, 55, 57, 3, 2]. The convection terms on all the grid levels are discretized by first-order upwinding, but on the finest grid, a source-term correction is applied which allows the second-order accurate central-difference solution, provided it is stable, to be recovered when the multigrid iterations converge. The fine grid discretized equations for one of the velocity components ϕ at iteration $n+1$, for a control volume, are set up in the form

$$\begin{aligned}[][a_P^{\text{u1}}\phi_P - a_{\text{E}}^{\text{u1}}\phi_{\text{E}} - a_{\text{W}}^{\text{u1}}\phi_{\text{W}} - a_{\text{N}}^{\text{u1}}\phi_{\text{N}} - a_{\text{S}}^{\text{u1}}\phi_{\text{S}} - b_P^{\text{u1}}]^{n+1} \\ = [r^{\text{u1}}]^{n+1} = [r^{\text{u1}} - r^{\text{ce}}]^n,\end{aligned} \tag{12}$$

where the superscripts u1 and ce denote first-order upwinding and central-differencing, n denotes the current values, r denotes equation residuals, b denotes source terms, and the coefficient subscripts denote the spatial coupling between the east, west, north, and south ϕ's. The second-order central-difference solution, $r^{\text{ce}} \to 0$, is recovered when $[r^{u1}]^{n+1}$ is approximately equal to $[r^{u1}]^n$.

The other approach is to use a stable second-order accurate scheme on all grid levels, in this case second-order

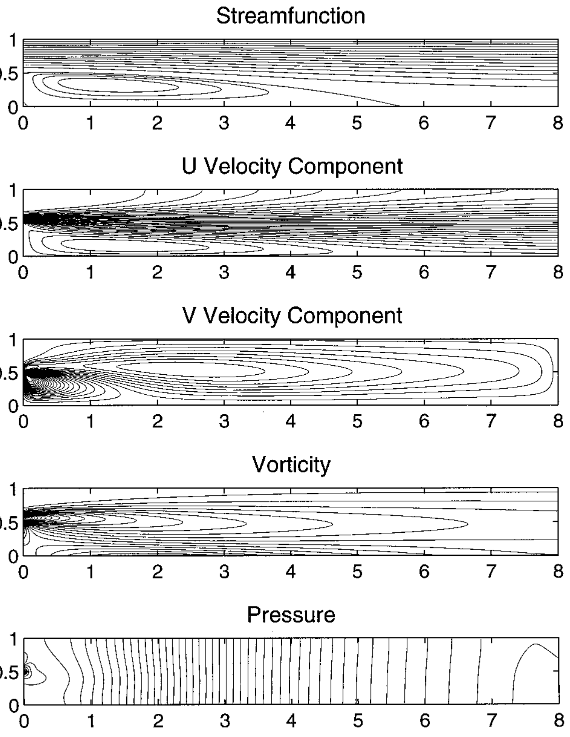


FIG. 9. Streamfunction, vorticity, pressure, and velocity component contours for Re = 300 symmetric backward-facing step flow, using the 2nd-order upwind convection scheme. The streamfunction contours are evenly spaced within the recirculation bubbles and in the interior of the flows, but this spacing is not the same. The actual velocities within the recirculation regions are relatively weak compared to the core flows.

upwinding. For the second-order upwind scheme, the equation coefficients are computed as in Ref. [48]. It is interesting to note that the cost of the momentum equation coefficient computations on the CM-5 using either second-order upwind or defect-correction is about the same and approximately 30% more than central-differencing. While the second-order upwind stencil involves two upwind neighbors and consequently has more communication, the defect-correction scheme has to compute more for the source terms. Shyy and Sun [47] compared the second-order upwinding approach with the approach of using first-order upwinding and central-differencing on all grid levels, for those cases in which the latter approach was stable. Comparable multigrid convergence rates were obtained for all three convection schemes in Re = 100 and Re = 1000 lid-driven cavity flows, whereas for single-grid computations there were relatively large differences in the convergence rates.

The restriction technique also affects the convergence rate. For finite-volume discretizations, *conservation* is the restriction procedure which is consistent with the finite-volume discretization. The reason is that the terms in the discrete equations represent integrals over an area (in 2-D). The method of integration for source terms determines the actual restriction procedure; we use piecewise constant. In either a staggered grid or cell-centered finite-volume formulation, the mass residual in a coarse-grid control volume is the sum of the mass residuals in the four fine-grid control volumes which comprise the coarse-grid control volume. In addition, the u-momentum equation residuals on the fine grid are treated as piecewise constant, and summed over the region corresponding to the coarse-grid u control volume under consideration. Due to the staggered grid, this involves summation over six fine-grid u control volumes, taking only half the contribution from four of the control volumes. The same procedure applies to the v-momentum equation residuals. For the prolongation step, we use a linear interpolation as described in [47] for staggered grids.

Typically the mass residuals are treated in this manner (conservation as the restriction procedure), but the summation of momentum residuals has proven problematic in other work [50, 47], even though it is physically consistent. By summing the mass residuals, and restricting u^h and v^h by cell-face averaging, satisfaction of the continuity equation can be identically maintained on the coarse grids at all times. This is not strictly necessary except at convergence, and it generates source terms in the coarse-grid momentum equations, since the cell-face averaged solutions will not in general satisfy the momentum equations. Thus, due to the source terms, the smoothing iterations may diverge when summation of residuals is used in conjunction with cell-face averaging of the velocity variables. In that case, cell-face averaging of the residuals may be better because, by effectively reducing (by $\frac{1}{4}$) the magnitude of the numerically derived source terms, convergence may be easier to obtain, albeit with a much slower convergence rate. Clearly, summation of both mass and momentum residuals is desirable, but what about the solution variables?

In the FAS formulation, the coarse-grid equations incorporate numerically derived source terms which are the difference between the restricted fine-grid equation residuals and the coarse-grid equation residuals based on an "initial" coarse-grid solution, which may be obtained by restricting the fine-grid variables. See Ref. [47] for details. In the original description of FAS, both solution variables and residuals were restricted [6]. However, the initial coarse-grid solution may also be taken from the previous multigrid cycle's "upstroke," i.e., the most recent solution, and we have observed that in some cases this strategy is preferable.

Figure 10 compares the convergence rates of V(3,2) cycles in two cases—curve 1 indicates cell-face averaging of the velocity variables and bilinear interpolation of pressure, while curve 2 indicates taking the most recent coarse-grid solution. All other parameters/procedures are identi-

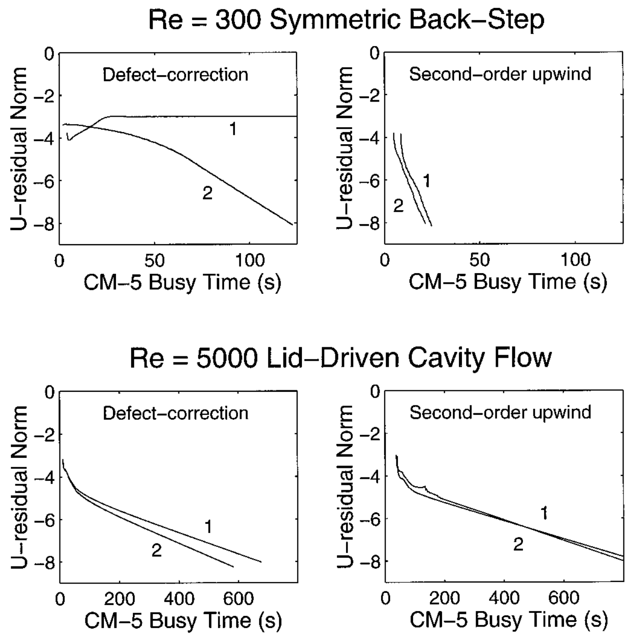


FIG. 10. The convergence path of the u-residual L_1-norm on the finest grid level in the 5-level Re = 300 symmetric backward-facing step flow and 7-level Re = 5000 lid-driven cavity flow. Contrasted are the two alternative treatments for restriction of solution variables: (1) cell-face averaging for velocities with bilinear interpolation for pressure; and (2) use of the most recent coarse-grid solution. In all cases the residuals are restricted by summation.

cal and are described below. The two approaches give comparable performance except in the step flow using defect-corrections, for which the solution does not converge (or diverge) when both solutions and residuals are restricted.

The problematic case indicates a competitive balance instead of a complementary effect, from the fine grid's perspective, between smoothing and coarse-grid correction. Consider the situation when the fine-grid mass and momentum residuals are everywhere zero, i.e., convergence. To maintain convergence in the case where the initial coarse-grid solution is taken from the previous cycle, the coarse-grid correction quantities Δu^{2h} as in

$$\Delta u^{2h}_{\text{to be prolongated}} = u^{2h}_{\text{smoothed, upstroke}} - u^{2h}_{\text{initial, downstroke}}, \quad (13)$$

where u refers to either velocity components or pressure and the subscripts are self-explanatory, must be zero. Since we are summing the fine-grid residuals, that contribution to the coarse-grid source term which comes from the fine grid is zero. However, the initial coarse-grid solution does not satisfy the governing equations discretized on the coarse grid; rather there *must be*, in the general case, nonzero source terms, since the coarse-grid solutions are an approximation to the fine-grid solution at corresponding locations. In the case of cell-face averaging, however, the coarse-grid continuity equation has no artificial source terms and so the pressure field is not able to adjust to reestablish the satisfaction of the momentum equations destroyed by the uncoordinated restrictions of u and v. If the coarse-grid source terms are important, as they generally are for the defect-correction cases since one is effectively switching between a central-difference and first-order upwind convection scheme, convergence may be impossible, as we observe in the step flow of Fig. 10. The key point is that for a system of coupled equations, the restriction of the solution variables must be coordinated with the restriction of the residuals. Our experience indicates that it is a more robust approach to let the initial coarse-grid values float, effectively letting the coarse-grid equations respond only to the approximation of the fine-grid residuals, which are obtained in a manner consistent with the finite-volume discretization of the governing equations. However, this technique should be investigated further.

Returning now to the truncation error criterion, we consider the step flow first. For this simulation, a 321 × 81 fine grid was used, with 5 multigrid levels. V(3,2) cycles were used, and for the smoothing iterations, 3, 3, and 9 inner iterations of the point-Jacobi type were taken. The relaxation factors were $\omega_{uv} = 0.6$ and $\omega_c = 0.4$. Figure 11 compares the convergence paths for different coarse-grid convergence tolerances in the FMG procedure, i.e., different initial solutions on the finest grid. The second-order upwind and defect-correction stabilization strategies are compared, also. The curves shown have the following meanings. "TE-1" and "TE-5" refer to the truncation error criterion described above, with the denominator set to 1 and 5. We have treated Eq. (6) as a heuristic to observe the sensitivity of this approach to the approximations that have been made. "1FMG" refers to the FMG cycle taking only one V cycle on each coarse-grid level, as shown in Fig. 7. "−3.0" and "−5.0" refer to constant coarse-grid convergence tolerances on each level; e.g., $\log_{10}\|r^h\| = -3.0$.

The "graded" tolerances, shown for the defect-correction scheme only, refer to our attempt to pick reasonable values for each level a priori. Specifying graded tolerances is equivalent to specifying a fixed tolerance using a residual that is normalized by a momentum flux instead of the number of control volumes, which is the approach used by Shyy and Sun [47]. Since, the equation residuals physically represent integrated quantities in the finite-volume formulation, the net residual regardless of the grid level should be roughly the same and hence graded tolerances are appropriate. However, the truncation error will not in general be spaced evenly, i.e., according to a factor of 4, because it depends on the solution, and so it becomes very difficult to choose a priori a good set of graded tolerances. Note also in the figure that CM-5 busy time is used since equiva-

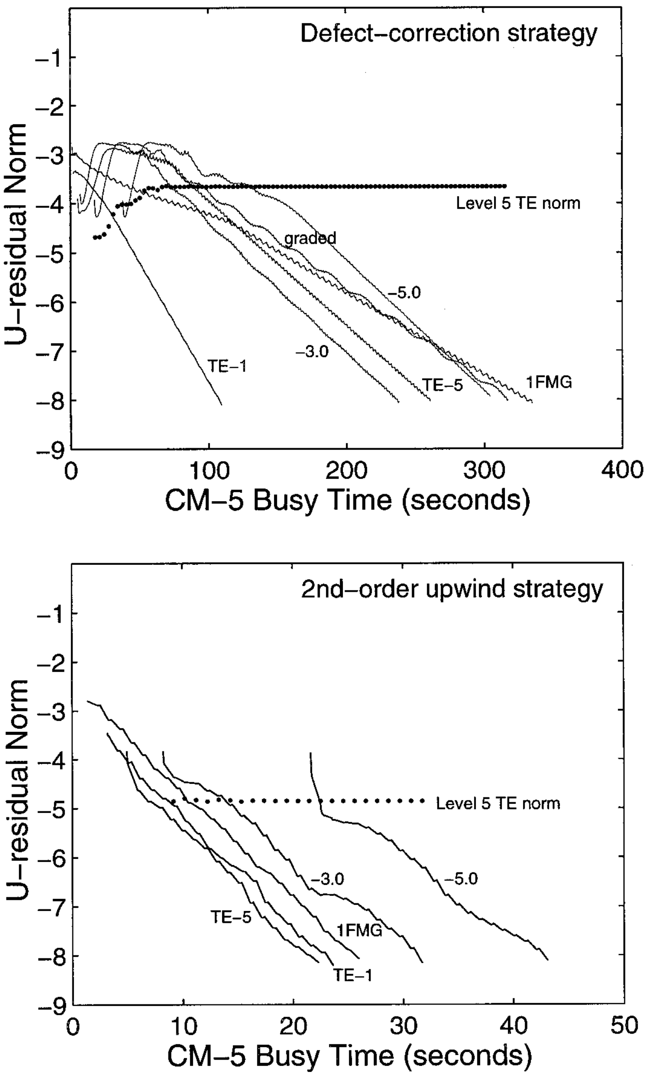


FIG. 11. The convergence path of the u-residual L_1-norm on the finest grid level in the 5-level Re = 300 symmetric backward-facing step flow. The relaxation factors used were ω_{uv} = 0.6, and ω_c = 0.4.

lent fine-grid iterations (work units) are not an accurate approximation for parallel computations. Some observations regarding Fig. 11 (the step flow) are:

(1) Second-order upwinding performs better than defect-correction. The u-residual norm reaches −8.0 in slightly more than 20 s on the CM-5, which equates to 140 work units, 20 fine-grid V(3,2) cycles. This corresponds to an amplification factor of 0.7 per cycle. Since the convergence is fast, the contribution of the FMG startup procedure to the overall cost is a significant fraction of the overall parallel run time, whereas it would not be on a serial computer.

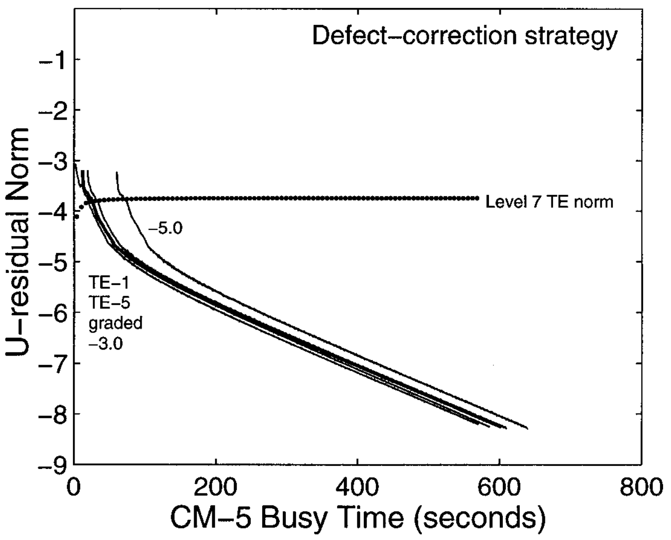
(2) The convergence rate for defect-correction depends strongly on the initial fine-grid guess. The best case is "TE-1" whose rate of error reduction per cycle corresponds to a smoothing rate of 0.95. As discussed earlier, in the defect-correction multigrid cycle we use first-order upwinding on all the coarse-grid levels, applying the source term corrections only on the finest grid. So the present results are consistent with the truncation error derivation sketched above which suggests a denominator 1 in the case of first-order discretizations.

It should be stressed that for each of the curves, identical procedures are used after the FMG procedure, i.e., once the fine-grid cycles are initiated, which is the point where the curves begin. Thus the fact that the asymptotic convergence rates differ reflects differences in the initial fine-grid guess. Brandt and Ta'asan [7] have shown that there can exist certain error modes in the initial fine-grid solution which are damped very slowly by the smoothing-correction multigrid combination when the flow is aligned with the grid and the convection terms are first-order upwinded, as in the present case; this may explain the observed convergence behavior.

For the lid-driven cavity simulation, a 321 × 321 fine grid was used, with 7 multigrid levels. Again, V(3,2) cycles were used, and for the smoothing iterations, 3, 3, and 9 inner iterations of the point-Jacobi type were taken. The relaxation factors were ω_{uv} = 0.5 and ω_c = 0.5. Some observations regarding Fig. 12 (the lid-driven cavity flow) are:

(1) The convergence rate for this recirculating-type flow, which corresponds to an error smoothing rate of about 0.99, is not as good as for the entering-type flow. However, the results appear to be consistent with the results obtained by Sockol [50]. For the Re = 5000 lid-driven cavity flow, using SIMPLE with $\nu_u = \nu_n = 1$ and $\nu_c = 4$ inner line-iterations and a W(1,1) multigrid cycle, Sockol found that 86 work units were needed to reach convergence (800 s on an Amdahl 5980). To reach a similar convergence tolerance, the present computation needed 30 cycles on the fine grid, which is 200 work units (64 s on the CM-5). Additional experiments have shown that for the cavity flow V(2,1) cycles are sufficient to obtain the same convergence rate as with V(3,2) cycles, and so our work units are effectively overstated by about 2.7 wu per cycle, yielding an equivalent of 119 work units.

Experience has shown that for this unique flow problem, the inner iterative method (line-relaxation or point-Jacobi) has no impact on the convergence rate of the outer iterations, and so this procedural difference between our work and Sockol's is irrelevant here. The slightly faster convergence observed by Sockol may be attributable to the use of a W cycle instead of a V cycle. However, for this 7-level problem size, W(1,1) cycles take twice as long as V(2,1) cycles on the CM-5, so the total run time is less using the latter.



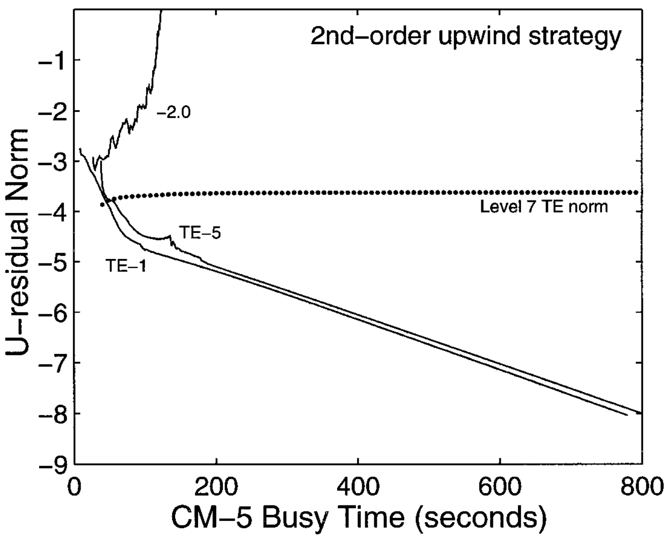
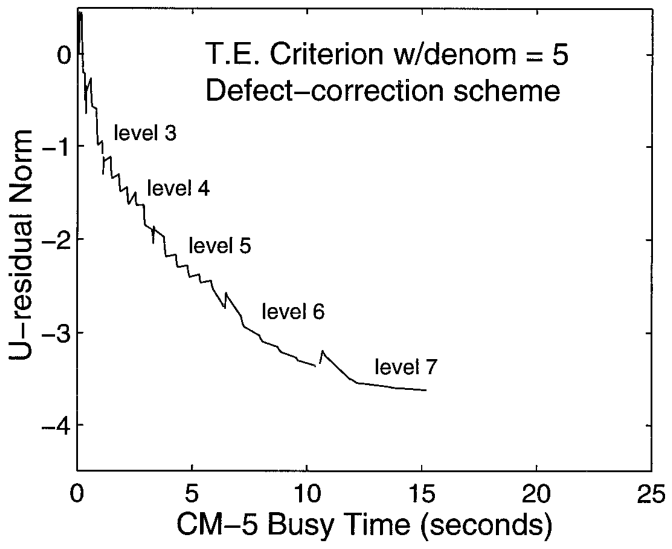


FIG. 12. The convergence path of the u-residual L_1-norm on the finest grid level in the 7-level Re = 5000 lid-driven cavity flow. The relaxation factors used were $\omega_{uv} = \omega_c = 0.5$.

(2) For either defect-correction or second-order upwinding, the convergence rate does not depend on the criterion used in the FMG procedure, i.e., on the initial fine-grid guess. In experiments with the defect-correction strategy, we have found that no matter how stringently the intermediate grid-level solutions are converged the error norm begins at approximately the same value, as shown in Fig. 13. The figure compares the convergence paths of the u-residual norm during the FMG procedure for the "TE-5" and "−3.0" curves, i.e., for the time preceding the starting time for the corresponding curves in the top plot of Fig. 12. These results indicate that the number of defect-correction iterations used on a given outer grid level during the FMG procedure does not seem to matter, an observation that is consistent with a convection-dominated flowfield—if the true velocity field has a strong upwind effect the defect-correction source terms are small.

In the second-order upwind cases of Fig. 12, convergence was only obtained for those cases which used the truncation error criterion. Experience with single-grid computations suggests that for high Reynolds number lid-driven cavity flows, the second-order upwind scheme is harder to converge than either the defect-correction, first-order upwind, or central-difference schemes for a given set of relaxation factors. Thus the second-order upwind cases which diverged can be made to converge by increasing the amount



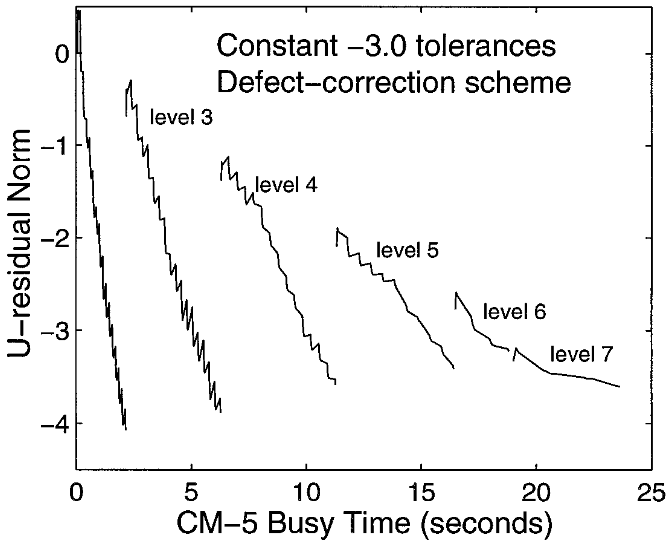


FIG. 13. The convergence path of the u-residual L_1-norm during the FMG procedure for the 7-level Re = 5000 cavity flow, using the defect-correction strategy, contrasting two criteria for controlling the coarse-grid cycling.

of smoothing and/or the relaxation factors. It is also possible that the bilinear interpolation prolongation procedure used here may be less compatable with a second-order upwinded solution than a central-differenced one.

4. CONCLUSIONS

Multigrid methods by their nature are iterative and, when the smoother is also iterative, as in the present case, they pose a difficult challenge for efficient data-parallel SIMD-style computation. The relative speed of front-end-to-processor and interprocessor communication, compared to the computation speed, is the key to efficiency. For V cycles, the coarse-grid contributions to the overall cost are nonnegligible but still small enough that nearly scalable performance is obtained as the problem size and number of processors are increased in proportion.

The single-grid SIMPLE method using the point-Jacobi solver ran at 420 MFlops on a 32-node (128 VU) CM-5. The multigrid cost per 7-level cycle was dominated by the smoothing costs, and the parallel efficiency was 0.65 compared to 0.8 for the single-grid program (about a 20% decrease). Parallel efficiency can be computed based on the detailed timing information as was done in Ref. [4]. Thus the speed of 7-level V cycles on 32 nodes is approximately 333 MFlops. The efficiency and speed improve slightly with fewer multigrid levels, but in the general case this comes at the expense of convergence rate.

The convergence rate of multigrid methods for the steady-state Navier–Stokes equations depends on many interacting factors, including the restriction/prolongation procedures, the amount of pre- and post-smoothing, the initial fine-grid guess, the stabilization strategy, and the flow problem. Furthermore, our experience with sequential pressure-based smoothers has been that none of these considerations are unimportant. V cycling in conjunction with the truncation-error-controlled nested iteration technique was found to be a robust method for both recirculating and entering type flow problems which gave convergence rates consistent and competitive with the results reported in the literature for steady-state incompressible Navier–Stokes equations using sequential pressure-based smoothers.

APPENDIX: ARRAY-BASED PARALLEL MULTIGRID STORAGE PROBLEM

For array-based parallel programming, the scalability of the multigrid storage cost is a problem. Generally, a variable number of multigrid levels is desired, but care must be taken not to waste memory. The naive approach is to make the multigrid levels explicit by adding a third dimension to all arrays, for example,

$$\text{REAL*8 A}(\,NI, NJ,\, k_{\max}),$$

where NI and NJ refer to the fine-grid dimensions for a 2-d problem, and $k_{\max}$ is the number of levels. Obviously the coarse grid levels should not be dimensioned to the fine grid extents, or else the size of problem which can be solved is greatly reduced. The total amount of memory used in the naive approach is the number of arrays, n_{array}, multiplied by the storage cost of each array,

$$\text{Storage} = NI\,NJ\,k_{\max}\,n_{\text{array}}. \tag{14}$$

The actual storage needed is only

$$\text{Storage} = \sum_{k=1}^{k_{\max}} NI_k NJ_k n_{\text{array}} = \sum_{k=1}^{k_{\max}} \frac{NI_{k_{\max}} NJ_{k_{\max}} n_{\text{array}}}{2^{(k_{\max}-k)}}. \tag{15}$$

The actual storage needed approaches only $(\frac{4}{3})\,NI_{k_{\max}}NJ_{k_{\max}}n_{\text{array}}$ as $k_{\max}$ increases. Thus the wasted storage is $(k_{\max} - \frac{4}{3})NI_{k_{\max}}NJ_{k_{\max}}n_{\text{array}}$ when the naive approach is used. Clearly this can become the dominating factor very quickly as the number of levels increases.

The resolution for serial computation [41] is to declare a 1-d array of sufficient size to hold all the data on all levels and to reshape it across subroutine boundaries, taking advantage of the fact that Fortran passes arrays by reference. One passes part of the 1-d array beginning at the appropriate element for the grid level under consideration and then locally dimensions it as a 2-d array with the proper dimension extents. This reshaping of arrays across subroutine boundaries is possible because the physical layout of the array is linear in the computer's memory.

For distributed memory data-parallel computation, however, the data arrays are not physically in a single processor memory, they are distributed among the processors. Instead of being passed by reference as is the case with Fortran on serial computers, data-parallel arrays are passed to subroutines by "descriptor" on the CM-5. The array descriptor is a front-end array containing 18 elements that describe the layout of the physical processor mesh, the virtual subgrid dimensions, the rank and type of the array, the name and so on.

In CM-Fortran the storage problem can be resolved using array "aliases." Array aliasing has a function similar to the Fortran EQUIVALENCE function. Storage is initially declared for all grid levels, explicitly referencing the physical layout of the processors. For example, an array A with fine-grid dimension extents $NI_{k_{\max}} \times NJ_{k_{\max}}$, is declared as follows for a 128-vector-unit CM-5 (32 nodes) with the vector units logically arranged in an 8×16 mesh:

$$\begin{gathered}\text{PARAMETER }(N_{\text{on-proc}} = (\tfrac{4}{3})NI_{k_{\max}}NJ_{k_{\max}}/(n_p^i * n_p^j),\\ n_p^i = 8,\, n_p^j = 16)\\ \text{REAL*8A}(N_{\text{on-proc}}, n_p^i,\, n_p^j).\end{gathered}$$

Actually, the factor $\frac{4}{3}$ should be increased slightly to account for "array padding" [54]. The array A is mapped to the processes using compiler directives so that the first dimension is laid out serially in each physical processor's memory, while the latter two dimensions are distributed across the 8×16 mesh of processors.

The second step is to then arrange for access to the parts of the A array corresponding to each grid level. For this purpose, array aliases (alternate front-end array descriptors for the same physical data) are created by compiler directives. For example, if the fine grid is 88×96, then an equivalent array descriptor to the "array section" A(1:88*96/128, 1:8, 1:16) is created which addresses the storage as an array A(88,96). With this trick arrays can be referenced inside subroutines as if they had the dimensions of the alias, with both dimensions parallelized. In CM-Fortran, specifically, a (:NEWS,:NEWS) layout of A(88,96) can be declared, even though in the calling routine the data come from an array of a different shape.

The array aliasing feature is relatively new in the CM-Fortran compiler evolution (Version 2.1-Beta [53]) and has not yet been implemented by MasPar in their compiler.[1] Previous multigrid algorithms in CM-Fortran were restricted to either the naive approach or explicit declaration of arrays on each level [13]. The latter approach is extremely tedious and leads to very large front-end executable codes, making front-end storage a concern. Thus, the present technique for getting around the multigrid storage problem, although requiring some programming diligence, is critical because it permits much larger multigrid computations to be attempted on parallel computers that use array-based parallel programming language. The large problem sizes, of the order of the largest possible problem sizes for the CM-5, are necessary to obtain good parallel efficiencies.

[1] Conversation with Larry Levine of MasPar customer support.

REFERENCES

1. M. Alef, *Parallel Comput.* **20**(10), 1547 (1994).
2. I. Altas and K. Burrage, *J. Comput. Phys.* **114,** 227 (1994).
3. E. L. Blosch, *Pressure-Based Methods on Single-Insruction Stream/Multiple-Data Stream Computers*, Ph.D. Thesis (University of Florida, Department of Aerospace Engineering, December 1994).
4. E. L. Blosch and W. Shyy, *Numer. Heat Transfer. B,* **26**(2), 115 (1994).
5. M. E. Braaten and W. Shyy, *Numer. Heat Transfer* **11,** 417 (1987).
6. A. Brandt, Multi-level adaptive solutions to boundary-value problems, *Math. Comput.* **31,** 333 (1977).
7. A. Brandt and S. Ta'asan, Multigrid solutions to quasi-elliptic schemes, in *Progress and Supercomputing in Computational Fluid Dynamics, Proceedings of U.S.-Israel Workshop, 1984,* edited by E. M. Murman and S. S. Abarbanel (Birkhäuser, Boston, 1985), p. 235.
8. A. Brandt and I. Yavneh, *J. Comput. Phys.* **101,** 151 (1992).
9. W. Briggs, *A Multigrid Tutorial* (SIAM, Philadelphia, 1987).
10. W. Briggs, *SIAM J. Sci. Comput.* **14**(2), 506 (1993).
11. T. F. Chan and R. S. Tuminaro, in *Parallel Computations and Their Impact on Mechanics, AMD-86,* edited by A. K. Noor (ASME, New York, 1988), p. 155.
12. A. J. Chorin, *Math. Comput.* **22**(106), 745 (1967).
13. J. E. Dendy, M. P. Ida, and J. M. Rutledge, A semicoarsening multigrid algorithm for SIMD machines, *SIAM J. Sci. Statist. Comput.* **13**(6), 1460, (1992).
14. J. P. Van Doormal and G. D. Raithby, *Numer. Heat Transfer* **7,** 147 (1984).
15. C. Douglas and J. Douglas, *SIAM J. Numer. Analy.* **30**(1), 136 (1993).
16. C. C. Douglas, Parallel multilevel and multigrid methods. Obtained from casper.cs.yale.edu via anonymous ftp.
17. T. A. Egolf, in *Parallel Computational Fluid Dynamics: Implementations and Results,* edited by Horst D. Simon (The MIT Press, Cambridge, MA, 1992), p. 27).
18. P. O. Frederickson and O. A. McBryan, Normalized convergence rates for the PSMG method, *SIAM J. Sci. Statist. Comput.* **12,** 221 (1981).
19. D. Gannon and J. Van Rosendale, *J. Parallel Distrib. Comput.* **3,** 106 (1986).
20. M. Griebel, in *Parallel Computational Fluid Dynamics '92,* edited by R. B. Pelz, A. Ecer, and J. Häuser (Elsevier, Amsterdam, 1993), p. 161.
21. S. N. Gupta, M. Zubair, and C. E. Grosch, *J. Sci. Comput* **7**(3), 263 (1992).
22. J. L. Gustafson, in *Proceedings of the Fifth Distributed Memory Computing Conference, Charleston, SC, 1990* (IEEE Comput. Soc. Press), p. 1255.
23. J. L. Gustafson, G. R. Montry, and R. E. Benner, *SIAM J. Sci. Statist. Comput.* **9**(4), 609 (1988).
24. P. J. Hatcher and M. J. Quinn, *Data-Parallel Programming on MIMD Computers* (MIT Press, Cambridge, MA, 1991).
25. B. R. Hutchinson and G. D. Raithby, *Numer. Heat Transfer* **9,** 511 (1986).
26. T. Hwang and I. Parsons, *Computers and Structures* **50**(3), 325 (1994).
27. D. C. Jespersen and C. Levit. A computational fluid dynamics algorithm on a massively parallel computer, *Int. J. Supercomputer Appl.* **3**(4), (1989).
28. V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing, Design and Analysis of Algorithms* (Benjamin/Cummings, Redwood City, CA, 1994).
29. V. Kumar and V. Singh, *J. Parallel Distrib. Comput.* **13,** 124 (1991).
30. J. Linden, G. Lonsdale, H. Ritzdorf, and A. Schüller, Block-structured multigrid for the Navier-Stokes equations: Experiences and scalability questions, in *Parallel Computational Fluid Dynamics '92,* edited by R. B. Pelz, A. Ecer, and J. Häuser (Elsevier, Amsterdam, p. 267.
31. J. Linden, G. Lonsdale, H. Ritzdorf, and A. Schuller, *Future Generation Computer Systems* **10**(4), 103 (1994).
32. J. Linden, G. Lonsdale, B. Steckel, and K. Stüben, Multigrid for the steady-state incompressible Navier-Stokes equations: A survey, in *International Conference for Numerical Methods in Fluids, Berlin, 1990* (Springer-Verlag), p. 57.
33. G. Lonsdale and A. Schüller, *Parallel Computing* **19**(1), 23 (1993).
34. P. Luchini and A. Dalascio, *Int. J. Numer. Methods in Fluids* **18**(5), 489 (1994).
35. O. A. McBryan, P. O. Frederickson, J. Linden, A. Schüller, K. Solchenbach, K. Stüben, C. A. Thole, and U. Trottenberg, *Impact Comput. Sci. Engr.* **3,** (1991).
36. P. Michielse, *Supercomputer* **10**(6), 10 (1993).

37. M. L. Minion, *Two Methods for the Study of Vortex Patch Evolution on Locally Refined Grids,* Technical Report LBL-35719, (Lawrence Berkeley Laboratory, Berkeley, May 1994).
38. A. Overman and J. Van Rosendale, Mapping robust parallel multigrid algorithms to scalable memory architectures, *Proceedings of the Third Copper Mountain Conference on Multigrid Methods,* edited by S. McCormick (Dekker, New York, 1993).
39. S. V. Patankar, *Numerical Heat Transfer and Fluid FLow* (Hemisphere, Washington, DC, 1980).
40. S. V. Patankar and D. B. Spalding, *Int. J. Heat Mass Transfer* **15,** 1787 (1972).
41. W. H. Press, S. A. Teukolsky, William T. Vetterling, and Brian P. Flannery, *Numerical Recipes in Fortran, The Art of Scientific Computing,* 2nd ed. (Cambridge Univ. Press, London, 1992).
42. C. M. Rhie, *AIAA J.* **27,** 1017 (1989).
43. P. Sathyamurthy and S. V. Patankar, *Numer. Heat Transfer, B* **25**(4), 375 (1994).
44. SC-NET: The Electronic Newsletter of the SIAM Supercomputing Activity Group (August 1995).
45. W. Shyy, *Computational Modeling for Fluid Flow and Interfacial Transport* (Elsevier, Amsterdam, *The Netherlands,* 1994).
46. W. Shyy, M.-H. Chen, and C.-S. Sun, *AIAA J.* **30,** 2660 (1992).
47. W. Shyy and C.-S. Sun, *Computers and Fluids* **22**(1), 51 (1993).
48. W. Shyy, S. Thakur, and J. Wright, Second-order upwind and central difference schemes for recirculating flow computation, *AIAA J.* **30,** 923 (1992).
49. R. A. Smith and A. Weiser, *SIAM J. Sci. Statist. Comput.* **13**(6), 1314 (1992).
50. P. M. Sockol, Multigrid solution of the Navier–Stokes equations on highly stretched grids with defect correction, in *Proceedings of the Third Copper Mountain Conference on Multigrid Methods,* edited by S. McCormick (Dekker, New York, 1993).
51. S. Thakur and W. Shyy, *Numer. Heat Transfer B* **23,** 175 (1993).
52. Thinking Machines Corporation, *CM-5 Technical Summary* (Cambridge, MA, November 1992).
53. Thinking Machines Corporation, *CM Fortran Release Notes, Preliminary Documentation for Version 2.1 Beta* 1 (Cambridge, MA, April 1993).
54. Thinking Machines Corporation, *Optimizing CM-Fortran Code on the CM-5* (Cambridge, MA, August 1993).
55. M. C. Thompson and J. H. Ferziger, *J. Comput. Phys.* **82,** 94 (1989).
56. R. S. Tuminaro and D. E. Womble, *SIAM J. Sci. Comput.* **14**(5), 1159 (1993).
57. S. P. Vanka, *J. Comput. Phys.* **65,** 138 (1986).
58. D. E. Womble and B.C. Young, Multigrid on massively-parallel computers, in *Proceedings of the Fifth Distributed Memory Computing Conference, Charleston, SC, 1990.* (IEEE Computer Society Press), p. 559.
59. J. Wright and W. Shyy, *J. Comput. Phys.* **107,** 225, (1994).
60. N. Wright and P. Gaskell, *Computers and Fluids* **24**(1), 63 (1995).
61. I. Yavneh, *SIAM J. Sci. Comput.* **14**(6), 1437 (1993).
62. S. Zeng and P. Wesseling, *SIAM J. Numer. Anal.* **31**(6), 1764 (1994).