NASA Contractor Report 187529
ICASE Report No. 91-20

ICASE

DOMAIN DECOMPOSITION METHODS IN COMPUTATIONAL FLUID DYNAMICS

William D. Gropp
David E. Keyes

Contract No. NAS1-18605
February 1991

Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, Virginia 23665-5225

Operated by the Universities Space Research Association

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665-5225

DOMAIN DECOMPOSITION METHODS IN COMPUTATIONAL FLUID DYNAMICS

William D. Gropp[1]
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439
and
David E. Keyes[2]
Department of Mechanical Engineering
Yale University
New Haven, CT 06520

ABSTRACT

The divide-and-conquer paradigm of iterative domain decomposition, or substructuring, has become a practical tool in computational fluid dynamics applications because of its flexibility in accommodating adaptive refinement through locally uniform (or quasi-uniform) grids, its ability to exploit multiple discretizations of the operator equations, and the modular pathway it provides towards parallelism. We illustrate these features on the classic model problem of flow over a backstep using Newton's method as the nonlinear iteration. Multiple discretizations (second-order in the operator and first-order in the preconditioner) and locally uniform mesh refinement pay dividends separately, and they can be combined synergistically. We include sample performance results from an Intel iPSC/860 hypercube implementation.

[1]The work of this author was supported in part by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under Contract W-31-109-Eng-38, `gropp@mcs.anl.gov`.

[2]The work of this author was supported in part by the NSF under contract ECS-8957475, by the IBM Corporation, and by the National Aeronautics and Space Administration under NASA Contract NAS1-18605 while the author was in residence at ICASE, NASA Langley Research Center, Hampton, VA 23665, `keyes@cs.yale.edu`.

1. Introduction.

The literature of computational fluid dynamics (CFD) ranges from elegant analyses of model systems to detailed analyses of realistic systems whose executions require hundreds of hours of supercomputer time. Software generally migrates from the former problem class to the latter at best slowly, and not without performance penalties, because the source of elegance, efficiency, or optimality is often the exploitation of special structure that is absent in applications. Therefore, the gap in attainable computational performance on ideal and practical CFD problems has little prospect of closing completely. Rather, since problems with less uniform structure usually are harder to map efficiently onto multiprocessors, parallel computing would appear only to widen the performance gap between the ideal and the real, while offering absolute improvements to both.

The combination of domain decomposition with preconditioned iterative methods extends the usefulness of numerical techniques for certain special partial differential equation problems to those of more general structure. The domains of problems with features inhibiting the global exploitation of optimal algorithms can often be decomposed into smaller subdomains of simpler structure on which extant solvers serve as local components of a parallelizable global approximate inverse. The computational advantages are usually sufficient to allow for the iteration required to enforce consistency at the artificially introduced subdomain boundaries, often even apart from parallelism. Size alone is often a sufficient advantage, since the computational complexity of many solution algorithms is a superlinear function of the discrete dimension, and thus p problems of size $\frac{n}{p}$ may be solved more cheaply than one of size n.

Iterative methods based on choosing the best solution in incrementally expandable subspaces allow the tailoring of computations to specified accuracy requirements. These methods can use multiple representations of the same underlying operator, ultimately converging in terms of a desired "high-quality" representation through a series of applications of the inverse of a "lower-quality" representation, called a preconditioner, that is cheap or parallelizable or possesses some other advantage. Though already useful in linear problems and on serial computers, the ability to operate with multiple representations of the operator proves even more significant in nonlinear problems and in parallel. In nonlinear problems, for instance, preconditioners for the Jacobian can be amortized over many Newton steps, while the solution is advanced through always up-to-date matrix-free approximations to Jacobian-vector products. In parallel, preconditioners can be constructed whose action requires less data exchange than a higher-quality representation would. One way to view domain decomposition is as a means of creating parallelizable preconditioners for iterative methods. The iteration required to piece together the solution at the artificial subdomain boundaries may be folded in with the iteration already implicit in the multiple levels of operator representation and, ultimately, with an outer nonlinear iteration as well.

Domain decomposition is a natural basis for partitioning programs across processors and partitioning data across memories, and allows a natural integration of local refinement, including refinements of mesh, of discretization order, or even of operator and the representation of the unknown fields. Though domain decomposition is as old as the analysis of engineering systems, the past decade has provided a significant theoretical foundation for model problems which has, in turn, provided heuristics for others. An aspect of interest to us is the migration in problem parameter space from the theoretically richly endowed "point" of the linear, selfadjoint problem for a scalar equation on a (quasi-)uniformly refined grid to the region of nonlinear multicomponent problems spawning a sequence of non-selfadjoint adaptively refined systems. Furthermore, we are interested in formulating such problems in a modular manner convenient to the design and maintenance of parallel software. For reasons of flexibility and inertia in the modeling of chemically reacting flows, in particular, we are primarily interested in finite difference or finite volume discretizations, but without relying on first-order methods since they are almost never competitive when the criterion is fewest operations for a given accuracy.

The philosophy of this paper has been set forth previously in [14] and [17], in which the gains of local refinement and multiple-order discretization, respectively, were illustrated. The backstep flow test problem with uniform mesh and discretization order was considered in [16], where it was shown that most of the portions of the code associated with the nonlinear and linear subtasks parallelize with comparable overhead. In this paper, we show that the confluence of these various tributaries leads to a conveniently programmed parallel implementation on medium-scale MIMD machines, and we explore its parallel efficiency on one such machine, the Intel iPSC/860. In the interest of brevity, we omit many algorithmic details covered in the references.

Section 2 describes a basic two-level algorithmic framework for implicitly discretized convection-diffusion systems. This is generalized in Section 3 to second-order adaptive refinements and placed in the context of an overall Newton iteration. The numerical results of Section 4 display the accuracy and parallel efficiency of some resulting combinations, and we conclude in Section 5 with a consideration of future objectives.

2. The Philosophy of Iterative Domain Decomposition. The domains of dependence of resolvents of elliptic operators, such as the spatial terms of the momentum and energy equations of (subsonic) fluid mechanics, are global, though there is a decay with the distance between the source and field points. The global dependence implies that data must travel across the grid from each point to all others during the solution process (for the satisfaction of sensible accuracy requirements). This requires a number of local data exchanges approximately equal to the discrete diameter of the grid or, possibly, a smaller number of longer-range exchanges derived from the use of multiple spatial scales. A length scale in between the integral length scale of the domain and the fine mesh parameter occupies a central place in our domain decomposition methodology. The intermediate scale need not directly determine the granularity of the parallelization, but it is convenient to base the parallel mesh data structure upon it.

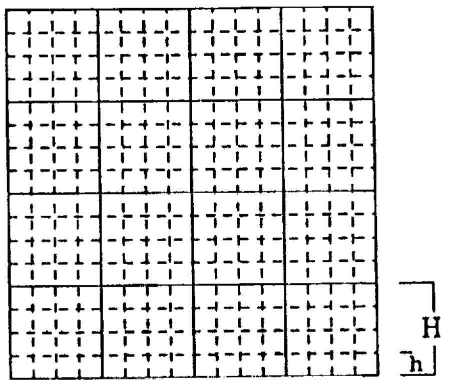

FIG. 1. *Schematic showing the length scales of the discretization, h, and decomposition, H.*

2.1. Global Data Transport. "Classical" results quantifying the trade-offs between purely local and global data transport are given in [2] and [8]. These papers show how preconditioned conjugate gradient iteration may be used to obtain solutions to two-dimensional selfadjoint elliptic problems in a number of iterations at most weakly dependent on the fine grid resolution through the logarithm of the ratio of the diameter of subdomains into which the global domain is divided, H, to the mesh parameter, h (see Figure 1). The cost in each case is the iterated solution of a subdomain vertex problem equivalent to a coarse discretization of the original operator with the subdomains as elements, along with the solution of the independent problems on the subdomains themselves (and on the one-dimensional interfaces in the case of nonoverlapping subdomains). Thus, the preconditioner is two-scale and requires regular non-nearest-neighbor data exchanges. For the preconditioner to be cost-effective, the nonlocal work should be subdominant. Practically, this requirement imposes a minimum H/h ratio. If the subdomain vertex solve in the preconditioner is replaced with a simple diagonal scaling, which removes the requirement of non-nearest-neighbor data exchanges, the bound on the iteration count rises in inverse proportion to H. If the subdomain solves themselves are likewise replaced with a simple diagonal scaling, it is a classical result for elliptic problems that the conjugate gradient iteration count rises in inverse proportion to h. The trade-off between the amount of work done in the preconditioner and the total number of iterations is thus well characterized, asymptotically.

The results for two-scale preconditioned selfadjoint problems have been extended in [4] and [5] to non-selfadjoint problems. Conjugate gradient iteration is replaced with the Generalized Minimum Residual (GMRES) method, and the bounds worsen by one or more powers of the factor $(1+\log(H/h))$. It is required in currently available convergence proofs that the coarse grid be sufficiently fine; in particular, a subdomain Reynolds number must be bounded. (Some convergence proofs for multigrid on non-selfadjoint problems avail themselves of a similar restriction.) Predecessors of the theoretically characterized non-selfadjoint form of the algorithm have been described for a scalar partial differential equation in [14] and [15]. These algorithms require more iterations of cheaper preconditioners and are roughly as effective (measured in execution time) as those possessing optimal convergence rates until H and h take on rather small values.

Whether the subdomains are assigned indivisibly to processors (as in our current codes), or whether the uniform tasks they represent are further subdivided in SIMD fashion, two-scale preconditioners significantly alleviate the sequential bottlenecks of global preconditioners such as incomplete factorizations. However, truly massive parallelism may require yet richer hierarchies of scales.

2.2. Preconditioned Krylov Iteration. Our domain-decomposed preconditioners are used in conjunction with the Krylov iterative method GMRES, described algorithmically in [23] and analyzed theoretically in [10] and [9] (in the equivalent form of the generalized conjugate residual method). Each iteration of GMRES involves a matrix-vector multiply requiring local data exchanges only and the preconditioner solve, in addition to some inner products. GMRES converges in a number of iterations proportional to the number of distinct clusters of one or more eigenvalues of the preconditioned operator. Loosely speaking, the greater the accuracy required, or the closer the cluster to the origin, the smaller the tolerance on what constitutes a single "cluster." Efficient use of GMRES in elliptic problems generally requires preconditioning to produce clustering. The appeal of GMRES is that it is robust and requires no user-estimated parameters. However, other iterative methods potentially requiring fewer inner products and smaller memory could be used instead; we mention the GMRES-Richardson hybrids in [19] and [24] and the Bi-CGSTAB method in [27] among contemporary candidates.

We summarize this section by establishing notation. A general framework for iterative domain decomposition methods for solving linearized elliptic systems consists of a global discrete operator, A; a global approximate inverse, B^{-1}; an iterative method requiring only the action of A and B^{-1}; and a geometry-based, contiguity-preserving partition of unknowns inducing a block structure on A and B.

We denote all subdomain vertices "cross-points." Ordering the interior points first, the interfaces connecting the cross-points next, and the cross-points last imposes the following outer tri-partition on the global discrete operator A:

$$A \equiv \begin{pmatrix} A_I & A_{IB} & A_{IC} \\ A_{BI} & A_B & A_{BC} \\ A_{CI} & A_{CB} & A_C \end{pmatrix}. \tag{1}$$

Note that the partitions vary greatly in size. If H is a quasi-uniform subdomain diameter and h a quasi-uniform fine mesh width, the discrete dimensions of A_I, A_B, and A_C are $\mathcal{O}(h^{-2})$, $\mathcal{O}(H^{-1}h^{-1})$, and $\mathcal{O}(H^{-2})$, respectively.

The structure of our preconditioner, B, is closely related to a conformally partitioned matrix

$$B = \begin{pmatrix} A_I & A_{IB} & A_{IC} \\ 0 & B_B & A_{BC} \\ 0 & 0 & B_C \end{pmatrix}, \tag{2}$$

consisting of the block-upper triangle of A, except for the replacement of A_C with an H-scale discretization of the original operator on the vertices, B_C, and the replacement of A_B with an h-scale discretization of the original operator along the interfaces of the decomposition with the normal derivative terms discarded, B_B. (See [6] for some numerical tests of this interface preconditioner.)

The application of B^{-1} to a vector $v = (v_I, v_B, v_C)^T$ consists of solving $Bw = v$ for $w = (w_I, w_B, w_C)^T$. It begins with a cross-point solve with B_C for w_C. This updates through A_{BC} the right-hand sides of a set of independent interface solves for subvectors of w_B and the right-hand sides of a set of independent interior solves for subvectors of w_I through A_{IC}. The interface solves, in turn, further update the right-hand sides of w_I through A_{IB}. Finally, the subdomain solves are performed. Note that the solves for w_B and w_I provide $\mathcal{O}(H^{-2})$-scale parallelism.

There is no dependence within the preconditioner of the cross-point or interface solutions upon the result of the interior solutions. This distinguishes the method from [2] and [5] and means that the $\mathcal{O}(h^{-2})$-sized block of the preconditioner is visited only once per iteration. However, an important variation of the preconditioner exists that represents a compromise between the strictly block triangular algorithm above and the cited methods. Following [2], we have found it advantageous to replace the right-hand side values v_C with weighted averages of the right-hand sides along adjacent interfaces before solving the cross-point system. This approach incorporates some lower-triangle coupling without any additional solves (see [14] for a detailed matrix interpretation).

3. Practical Domain Decomposition Algorithms for CFD.

In the present contribution we merge four tributaries of our recent work: (1) local uniform mesh refinement, (2) use of a pair (A, B) in which B is of lower order (defect correction), (3) nonlinear solvers, and (4) implementation on parallel processors.

3.1. Locally Uniform Mesh Refinement.

In many cases, the problems generating the discrete systems to be solved by domain decomposition have several different physical length scales. Since the polynomial approximations underlying local finite discretization methods are length-scale specific in their validity, mesh refinement (perhaps in combination with refinement of discretization order) is often used to produce an accurate solution. Locally uniform mesh refinement [14] is an adaptive resolution technique that is well suited to domain decomposition. By it, rectangular subdomains are refined with locally computationally regular tensor-product meshes. This refinement permits easy and efficient vectorization and allows consideration of fast solvers as components of domain-decomposed preconditioners. Different subdomains may have different mesh refinement, but the refinement is of a uniform scale within a single subdomain. This regularity allows a concisely expressed and flexible algorithm. Changes in grid refinement at interfaces between subdomains are accommodated with mutually overlapping phantom points and biquadratic interpolation. The phantom points allow the use of conventional finite-difference techniques (for second-order differential operators) in generating the difference equations at the

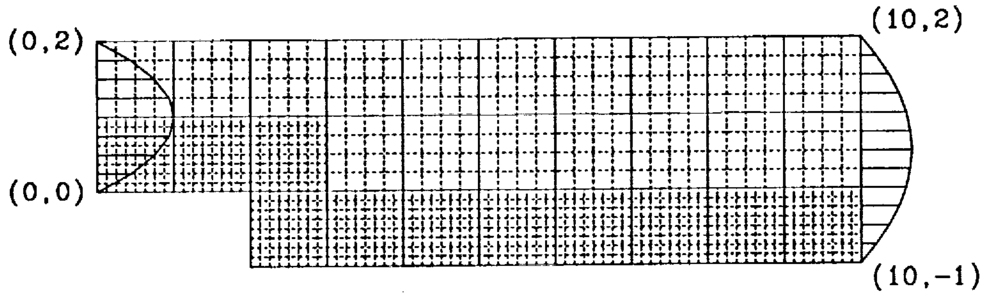

FIG. 2. *Schematic of a composite grid for the backstep flow problem, with well-developed inflow (left) and outflow (right) velocity profiles superposed. The upper and lower surfaces are rigid walls. Refinement is employed near the step and in the recirculation region. (The composite grids actually used to generate the data in the following section are finer than shown here.)*

subdomain interfaces. The selection of general refinement criteria is well examined in the literature (see, e.g., [28] for a recent review) and beyond the scope of the present contribution. In the specific example presented below, a sufficient refinement strategy is suggested by the known location of the vorticity singularity and confirmed by the ability to accurately reproduce known results. Much more efficient refinement strategies exist and we plan to incorporate them in a self-adaptive way in the future.

In [14], the classic problem of Poisson's equation in an L-shaped domain was used to illustrate the memory and execution time savings allowed by subdomain-based local mesh refinement over global refinement, without sacrifice of accuracy. For an effective resolution of $h^{-1} = 128$, for instance, a reduction factor of just over 6 in execution time accompanied a reduction factor of just over 5 in the number of unknowns required to represent the solution. Empirical observation of iteration counts in the globally and locally refined cases suggests that it is the finest mesh spacing, not the number of unknowns *per se*, that determines the convergence rate in variably refined domain decomposition algorithms. Though the theory developed for quasi-uniform grids cited in Section 2.1 is not directly applicable to our tests, the results of the tests and the theoretical estimates are consistent if the maximum H/h, i.e., the discrete dimension of the finest tile, is employed in the latter.

Figure 2 below illustrates how the locally uniform refinement technique is applied to the L-shaped backstep flow problem.

3.2. Accelerated Defect Correction. A conventional defect correction method for solving the system of equations

$$N(u) = 0, \tag{3}$$

where N depends continuously on u, is as follows. We suppose that we can easily solve a related problem

$$\bar{N}(u) = f.$$

Then we initialize u by solving

$$\bar{N}(u^0) = 0$$

and iterate:

$$\bar{N}(u^{k+1}) = \bar{N}(u^k) - N(u^k).$$

If the iterations converge, they converge to a solution of (3). In our context, $\bar{N}$ is simply a lower-order discretization of N.

For linear $N(u)$, this stationary defect correction can be accelerated by using the $\bar{N}$ discretization as the basis for a domain-decomposed preconditioner B for $A \equiv N$. In [17] we found an accelerated version of defect correction to be useful in maintaining second-order accuracy in a CFD finite-difference discretization while employing only the more convenient first-order upwind differencing for the convective terms in the preconditioner. Full second-order-in-h truncation error convergence was observed for smooth problems. Two types of measurements were made to quantify the performance of this algorithm. For a fixed h, the number of iterations required for algebraic convergence of the preconditioned GMRES method was compared with a case in which A and B were based on the same first-order upwind discretization. The method with second-order A required more iterations, but never more than 1.5 times as many. In terms of the execution time required to achieve a fixed truncation error, the method with second-order A was an order of magnitude more efficient because of its sparser grid.

3.3. Newton's Method. For the solution of steady reacting flow problems, robust variations of Newton's method, assisted as necessary by parameter continuation, are often preferable to less fully coupled iterative methods or associated explicit time-marching methods (see, e.g., [25]). We regard the current work as a prelude to building reacting flow codes for MIMD parallel architectures; thus, it is natural to focus on Newton methods.

We write the overall system in the form

$$F(\phi) = 0, \tag{4}$$

where ϕ represents a column vector of all of the unknowns. Equation (4) may be solved efficiently by a damped modified Newton method provided that an initial iterate $\phi^{(0)}$ sufficiently close to the solution ϕ^* is supplied. The damped modified Newton iteration is given by

$$\phi^{(k+1)} = \phi^{(k)} + \lambda^{(k)} \delta\phi^{(k)}, \tag{5}$$

where

$$\delta\phi^{(k)} = -(\tilde{J}^{(k)})^{-1} F(\phi^{(k)}), \tag{6}$$

where the matrix $\tilde{J}^{(k)}$ is an approximation to the actual Jacobian matrix evaluated at the k^{th} iterate. We refer to $\delta\phi^{(k)}$ as the k^{th} update. When $\lambda^{(k)} = 1$ and $\tilde{J}^{(k)} = J^{(k)} \equiv \frac{\partial F}{\partial \phi}(\phi^{(k)})$, for all k, a pure Newton method is obtained. The iteration terminates when some (scaled) 2-norm of $\delta\phi^{(k)}$ drops below a given tolerance. In well-conditioned systems, this will, of course, also be true of the norm of $F(\phi^{(k)})$.

From the discussion of equations (5) and (6) we identify the five basic tasks that together account for almost all of the execution time required by the Newton algorithm: (1) DAXPY vector arithmetic, (2) the evaluation of residual vectors, (3) the evaluation of Jacobians, (4) the evaluation of norms, and (5) the solution of linear equations involving the Jacobian matrix. The DAXPY requires no data exchanges between neighboring points. The residual and Jacobian evaluation (performed analytically here) require only nearest-neighbor data exchanges. The evaluation of norms and the linear system solution require global data exchanges and are hence the focus of a parallel implementation. In a general-purpose Newton algorithm, significant amounts of code must be written beyond the steps listed here. Automating the continuation, damping, and Jacobian re-evaluation strategies can greatly affect the efficiency of a Newton method. However, these essential additional tasks require insignificant amounts of computational work not already in the five categories above.

3.4. Parallel Implementation. Preceding sections have described a convenient domain-based clustering of work into "tiles" while flagging the phases of the overall algorithm that require inter-tile data exchanges. A parallel implementation follows directly, except for decisions regarding the solution of the global coarse grid problem, for which the best algorithm is architecture- and problem-dependent. Many details of serial, parallel shared-memory, and parallel distributed-memory domain decomposition algorithms for linear problems have been given in [13] and [15]. It is interesting that "good" algorithms for all three computing environments can share over 95% code in common.

Work arrays for the data structures associated with each tile are allocated individually to available processors according to heuristic load-balance criteria, without priority concern for proximity in the processor network of processes associated with neighboring subdomains. (Users of domain decomposition algorithms on earlier Intel hypercubes concluded that the penalty for failing to preserve nearest-neighbor connections in subdomain-to-processor mappings is at most 20% percent in total runtime [11]. This is non-negligible, but worst-case load imbalance penalties when nearest-neighbor connections are slavishly preserved can be arbitrarily higher. Mapping algorithms simultaneously satisfying good load-balance and good subdomain-processor locality constitute an on-going research effort. From a practical point of view, a cost-benefit analysis of the mapping algorithm itself must be taken into consideration. For representative pointers into this literature, [1] and [20] may be consulted.) A buffer is maintained around the perimeter of each tile of a width corresponding to the semibandwidth of the difference stencil in use on that tile. These buffers are refreshed by interpolation from neighboring tile interiors at appropriate synchronization points.

Generally, individual processors are responsible for multiple subdomains, and tiles assigned to the same processor are processed sequentially within each synchronized phase of the algorithm. Optimizations have been incorporated into the parallel code to packetize data exchanges between the same processors resulting from different tile-tile interfaces. On a machine where interprocessor communication is relatively expensive, such as the iPSC/860, message buffering is potentially valuable, but more attention to the tile-processor mapping is required to fully exploit it. The major uses of the freedom of MIMD (as opposed to SIMD) programming are in the variable resolution of tiles (for adaptive discretization), the variable number of tiles per processor (for load balance), and the enforcement of boundary conditions. Boundary conditions are often a bugaboo of parallel programming, but we must recognize them only in the preconditioner and only in an approximate manner. This is because local boundary conditions of any mathematically reasonable type can be cast in the form of matrix-vector multiplies with the operator A.

It is typically uninviting to solve the relatively small preconditioner coarse grid problem defined by the tile vertices, a sparse linear system, in a distributed fashion. There is too little arithmetic work per processor at modest tile-to-processor ratios. Neither is it optimal to gather the distributed right-hand side data for this problem onto a single processor, solve it sequentially while the other processors wait, and scatter the result back. The communication time of the latter approach can be cut roughly in half by broadcasting the right-hand side data to all processors and solving redundantly on each. The redundant coarse grid solution is used in generating the parallel performance data given below.

A different technique, called the "asynchronous crosspoint solve," allows the inversion of the diagonal blocks of B_B and A_I in the preconditioner to begin before the coarse grid solution has completed. Since the result of the preconditioner solve is linear in the components of the right-hand side, it is possible to compute in a preprocessing step the discrete Green's functions associated with each vertex. Storing these Green's functions requires four extra vectors of the dimension of the number of unknowns in the discretization for each unknown field in the system of governing PDEs. (Thus, for example, a two-component streamfunction-vorticity system requires 8 extra vectors of size $2N$, where the composite grid consists of N points.) After the coarse grid solve is completed, its high communication requirements overlapped with the bulk of the preconditioner solve, the proper components of the vertex Green's functions can be added in. The vertex Green's functions would generally have to be recomputed each time the Jacobian was re-evaluated, at the cost of four sets of subdomain solves. The optimal tradeoff between the potentially inhomogeneous workload and extra preprocessing and storage of the Green's function method versus the parallel inefficiency of solving the vertex problem is both architecture- and problem-specific, and has not been pursued in the current code.

4. Flow over a Backstep.

We illustrate the capabilities of the nonlinear domain-decomposed solver on a classic model problem from computational fluid dynamics, the flow over a backstep, studying both solution accuracy as a function of discretization and parallel performance as a function of refinement and processor granularity.

Though it is a favorite demonstration problem, there is no single canonical backstep flow configuration in the literature. The principal variations lie in the choice of symmetric channel geometry or a flat wall opposite the step, in the characterization (plug flow, fully developed, or experimentally measured) of the upstream boundary conditions, in the ratio of step height to channel width, and in the smoothness of the step itself. For present purposes, we fix these choices as a flat opposite wall, a fully developed inlet profile (located two step heights upstream), and a channel expansion ratio of 2 to 3 occurring abruptly at the step (see Figure 2).

Inasmuch as the flow is well characterized as laminar, steady, and two dimensional in the Reynolds number range we model, we use the streamfunction-vorticity formulation of the incompressible Navier-Stokes equations, in which velocity components (u, v) are replaced with (ψ, ω) through

$$u = \frac{\partial \psi}{\partial y}, \; v = -\frac{\partial \psi}{\partial x}, \text{ and } \omega = \frac{\partial u}{\partial y} - \frac{\partial v}{\partial x}. \tag{7}$$

The streamfunction satisfies the Poisson equation

$$-\nabla^2 \psi + \omega = 0, \tag{8}$$

and the vorticity the convection-diffusion equation

$$\text{Re}\left[u\frac{\partial \omega}{\partial x} + v\frac{\partial \omega}{\partial y}\right] - \nabla^2 \omega = 0. \tag{9}$$

This system is nondimensionalized, with the step height as the reference length and the centerline inlet velocity as the reference velocity. (Some authors employ the *mean* inlet velocity in nondimensionalizing. Their Reynolds number, Re, is thus two-thirds the size we report, for the equivalent flow configuration.)

We observe that (apart from boundary conditions), the Jacobian of this system has the form

$$J = \begin{pmatrix} -\nabla^2 & I \\ C_1 & -\nabla^2 + C_2 \end{pmatrix}, \tag{10}$$

in which matrices C_1 and C_2 approach zero with the Reynolds number. If convenient boundary conditions could be specified for the vorticity, a good preconditioner for this system could comprise a pair of fast Poisson solvers, but this condition is typically unmet in practice.

The boundary conditions employed in the numerical tests are specified with reference to the domain in Figure 2. The inlet streamfunction and vorticity are derived from integration and differentiation, respectively, of the assumed well-developed upstream velocity profile, $u(0,y) = y(2-y)$ and $v(0,y) = 0$. (ψ is referenced to zero at the origin of coordinates.) Along the fixed, impenetrable no-slip upper and lower walls ψ is constant; hence all its tangential derivatives are zero. Through equation (8), ω is thus set equal to $-\frac{\partial^2\psi}{\partial n^2}$, where n is the unit normal, chosen in the vertical by default at the degenerate corner of the step. (Numerical experiments with alternative choices described in [22] did not suggest an obvious preferred way of breaking this degeneracy, the mathematical artifact of an infinitely sharp step, and it is evident in the results that our arbitrary choice is not limiting as regards the phenomena of interest.) Finally, along the outflow boundary we used extrapolation conditions: $\frac{\partial\psi}{\partial x} = 0$ and $\frac{\partial^2\psi}{\partial x^2} = 0$. These conditions *were* accuracy-limiting at sufficiently large Reynolds number in a straightforwardly removable way, as described below.

We employed a variety of discretizations at seven Reynolds numbers spanning the range from 50 to 200 in increments of 25. We ran the full set of problems on a Sparcstation-1, then ran a subset of problems at Reynolds number 100 on the Intel iPSC/860, varying the number of processors employed from the smallest number containing sufficient aggregate memory up to the maximum available (32) in order to evaluate performance. We employed zeroth-order continuation to shorten the time required to sweep through Reynolds number space, that is, we used the solution at the next lower Reynolds number as a starting estimate at the current, beginning with the case (Re = 50) in which the nonlinear influence is the smallest. Continuation is often employed in nonlinear solvers for robustness, but in this Reynolds range we employed it only for convenience. In no case did the Newton algorithm suffer convergence difficulty in starting from "cold" estimates obtained either by extrapolating the inlet flow unchanged downstream and patching it to an initially stagnant region behind the step or by assuming the entire domain to be stagnant.

A sample solution at Re = 100 is contoured in Figure 3. The dividing streamfunction contour lies slightly below the top of the step, towards which it climbs from a pure Stokes (Re = 0) solution, reproducing a known feature of this flow field. The center of the channel is vorticity-free. The vorticity is high on either side of the channel just upstream of the step, and the highest vorticities occur in the neighborhood of the step itself, where it is undefined. It is evident from the figure that the flow returns to an almost symmetrical shape following the aysmmetrical expansion, though the exit profile has not yet achieved its asymptotic parabolic profile only eight step heights downstream.

4.1. Solution Accuracy. Since no exact solutions of the backstep flow problem are available, we rely on comparisons of functionals of the solution obtained previously by other investigators in evaluating the accuracy of our numerical solutions. Four such scalar functionals are the length of the recirculation zone (as defined by the reattachment point of the dividing streamfunction contour), the strength of the recirculation (as defined by the maximum magnitude of the streamfunction in the recirculation region), and the downstream and transverse coordinates of the point at

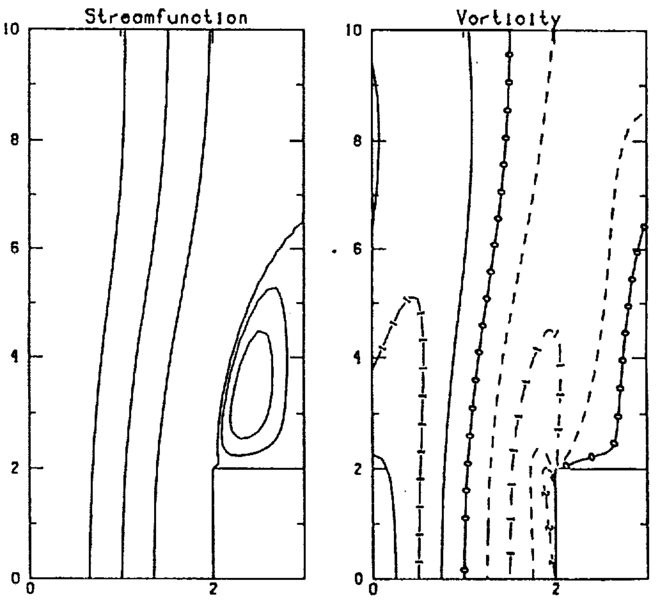

FIG. 3. *Contour plots of streamfunction and vorticity for Reynolds number 100 flow over a backstep. (In the display orientation the flow is from bottom to top, and the true aspect ratio is distorted, allowing more detail in the transverse direction.)*

which the maximum magnitude of the streamfunction is achieved.

For Reynolds numbers in the range investigated (50 to 200), the length of the recirculation zone is well approximated as a linear function of Re. (See [21] which discusses a similar study with the spectral element method and supplies references to earlier experimental and numerical investigations.) We adopt the notation L_r for this length (measured in step heights) and show in Figure 4 previously obtained results for L_r versus Re, along with results of our domain decomposition code. The spectral results of [21] and the (evidently highly resolved) finite difference results of [7] on domains sufficiently extended in the downstream direction fall very tightly around the dashed line connecting Re = 50, L_r = 2.87 with Re = 200, L_r = 8.18. (At higher Reynolds numbers, the time-averaged reattachment length is known to slow as a function of Re, achieve a maximum, and eventually retreat part way upstream, though this behavior occurs in the turbulent regime.) Note that the reattachment point at Re = 200 lies a bit beyond the edge (dashed cutoff) of the domain of Figure 2. Because an accurate L_r is unmeasurable in this case, some data points are missing at Reynolds number 200. The two data points shown at Re = 200 correspond to discretizations that are artificially diffusive enough to severely shrink the recirculation zone. The close approach of L_r to the boundary at Re = 175 allows showing the manner in which the extrapolative downstream boundary condition fails by pulling the tail of the recirculation zone out of the domain. The less artificially diffusive the discretization, the greater the effect of the outflow boundary condition on L_r.

As listed in the legend of Figure 4, first- and second-order upwinding are combined with resolutions of ten gridpoints per unit length (base) and twice and thrice this resolution in refined regions near and downstream of the step. It is observed that

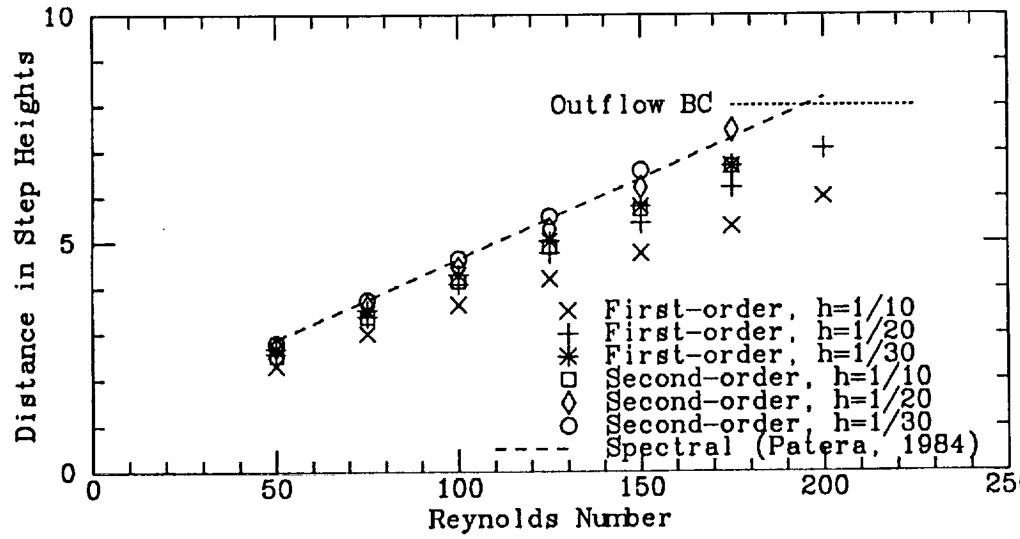

FIG. 4. ***Reattachment length versus Reynolds number for six different combinations of mesh refinement and discretization order for the backstep flow problem.***

switching from first- to second-order discretization is more effective than adaptive h-type refinement, and that a combination of second-order and modest refinement achieves nearly full accuracy for the Reynolds numbers considered. Using accepted values of the reattachment length at various Reynolds numbers to define errors, the first-order discretizations plainly yield first-order convergence, whereas the second-order discretizations yield superlinear convergence, full quadratic convergence being difficult to measure with just three points.

The maximum magnitude of the streamfunction in the recirculation zone normalized by the difference in streamfunction across the entire channel, $|\Delta\psi|_{max}/\psi_0$, is known to approach from below a value of approximately 2% as the Reynolds number increases through our range of interest. The broken line in Figure 5 closely fits the data of [21] in the range shown and the markers show how the values of recirculation strength are approached under the same set of six discretization combinations tested above. For this rather sensitive functional, doubling resolution is more effective than doubling order relative to the crudest approximation. The fundamental problem of upwind differencing in the presence of recirculation is discussed in [3] and references therein. Fortunately, recirculation occurs in flow regions where the Reynolds number based on the local velocity is small in typical applications. In such regions, second-order central differencing in A poses no problems for the upwind-preconditioned system [17], and the local discretization can be adaptively switched. However, this adaptive switching is not yet incorporated.

Throughout the middle of the Reynolds number range, the downstream coordinate of the point of maximum recirculation streamfunction relative to the edge of the step and normalized by overall recirculation zone length, $\Delta x_m/L_r$, is 0.3 ± 0.01. The corresponding transverse coordinate, Δy_m, is 0.4 ± 0.03 step heights, nearly

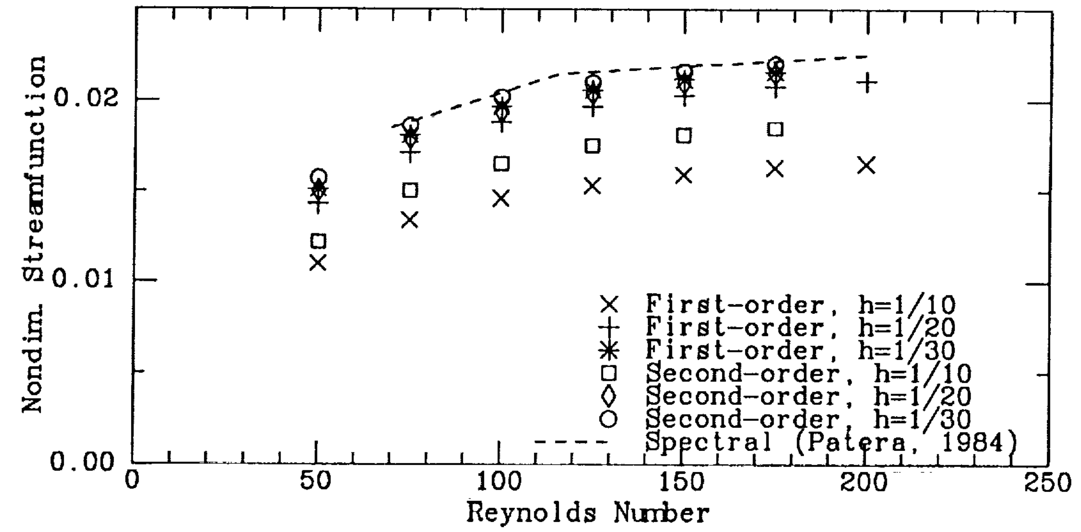

FIG. 5. *Maximum normalized recirculation streamfunction versus Reynolds Number for six different combinations of mesh refinement and discretization order for the backstep flow problem.*

independent of Re. These are both in close agreement with earlier numerical and experimental results.

From the graphs it is clear that a first-order upwind method is an inefficient means of obtaining accurate solutions as the Reynolds number increases; but that it nevertheless makes a convenient preconditioner for a higher- (here, second-)order upwind operator. These conclusions are not new; the novel aspect of this work is the modular manner in which the composite grid preconditioned operator is constructed which leads to convenient local refinement and parallelism. The selection of which tiles to refine and how much to refine them was crudely guided by knowledge of the problem, but is clearly amenable to finer tuning through automatic error estimation. Perhaps the worst aspect of the performance of the less accurate methods is that they fail to detect that the domain is too short at the highest Reynolds numbers tested, because of their artificially high diffusivity. Complaints that heavily upwinded discretizations conceal their own errors are common in the literature and are among the strongest incentives for building modular software that makes checking alternative discretizations and refinements feasible and convenient [18].

Counting streamfunction and vorticity values separately, there are 5,862 degrees of freedom in the base grid, 10,422 in the intermediate, and 21,702 in the most refined problem. We emphasize that these are far from competitive refinements for tile-based finite-difference discretizations, since the streamwise direction is very overresolved relative to the transverse when grid elements are squares, as here. Nonisotropic tessellations and nonisotropic refinement of individual tiles are clearly possibilities that fit comfortably within the tile framework. The data above serve only to show how refining locally and changing operator order may be done without sacrificing regularity in the implicit portions of the computation. Though not obtained on optimal discretizations, the data make positive statements about the discrete solution process. For convective-diffusive problems in which geometrical complexities in the boundary and the distribution of sources or sinks require large numbers of unknowns, the fully implicit, fully nonlinear iterative solver performs robustly.

4.2. Convergence Behavior. We comment briefly on several aspects of the nonlinear and linear algebraic convergence in the numerical experiments reported here.

The effectiveness of the continuation procedure can be communicated by means of a typical example from the "middle" of the physical and numerical parameter space explored in this study. We consider the Re $= 100$ flow with a second-order upwinded operator and one level of refinement from a base grid consisting of 2 tiles per unit length (the step height) and 5 mesh intervals per tile, resulting in an effective h^{-1} of 20 in the refined regions. The following timings are quoted from a Sparcstation-1. From a "cold" start, with an initial nonlinear residual Euclidean norm of 2.13×10^1, four Newton steps were required to drop the final residual to 9.96×10^{-3}. These four Newton steps required a total of 132 preconditioned GMRES iterations (with a different first-order upwind domain-decomposed preconditioner for each of the four sets) and a total of 229 sec of CPU time. From a "warm" start consisting of the converged solution to the problem at Re $= 75$ and an initial nonlinear residual Euclidean norm of only 4.25×10^{-1}, three Newton steps brought the final residual to a comparable 9.36×10^{-3}. The last Newton step required only one preconditioned GMRES iteration, thus the construction of the preconditioner for the third stage was largely unamortized effort; nevertheless the totals of 63 GMRES iterations and 103 sec of CPU time represented a little less than half the effort of the "cold" start case.

The relatively modest relative reductions in nonlinear residual ($\mathcal{O}(10^3)$) at which convergence was declared were sufficient to bring out the full truncation error potentials of the discretizations employed. To evaluate this, we ran a second "cold" case until the final nonlinear residual was 9.85×10^{-6}, or three additional orders of magnitude. This required three additional Newton steps, for a total of seven, and totals of 254 preconditioned GMRES iterations and 463 CPU seconds. No differences were observed in any of the functionals plotted in the previous subsection. Thus, the nearly doubled numbers of GMRES steps and CPU cycles were unnecessary from a "bottom-line" viewpoint.

As can be gathered from the comparison of the just cited "cold" runs, terminated at different stages, we observe a Newton convergence (as monitored by the residual, since the exact solution is not known) which is closer to linear than to quadratic. It is difficult to estimate how much of the convergence history is spent in the domain of quadratic convergence of Newton's method in these problems, but we do not expect to see full quadratic convergence because we employ an inexact Newton method; that is, we tune the convergence of the linear system solves at each Newton step to the outer progress, with a mixed relative-absolute tolerance. Further experimentation may yield better couplings of inner to outer iterations for this class of problem, but for the precision with which we report relevant functionals of the overall solution in this investigation, the asymptotic convergence rate of Newton's method is not a crucial feature.

We noticed two interesting couplings of the convergence progress of the backstep problem to the discretization technique. Considering first the discretization order, we found that the first-order discretization required more Newton steps of fewer GMRES iterations each than the second-order discretization on the same grid to achieve a given level of nonlinear residual reduction. Rather than four Newton steps comprising 132 GMRES iterations in the short "cold" start case discussed above, a first-order discretization of the same problem required six Newton steps comprising a total of 123 GMRES iterations. The final nonlinear algebraic residual was a comparable 9.78×10^{-3}, and the CPU time required was only 149 seconds instead of 229 seconds. Since the approximation to the underlying differential equation was demonstrably superior for the second-order discretization, the extra 54% of CPU cycles was well worth it, but the difference in algebraic behaviors of the two discretizations is interesting to note. It suggests the hypothesis that a push to higher-order upwind discretizations would eventually be defeated by the rising cost of solving the resulting discrete equations. The cross-over point remains to be determined and should be evaluated on the basis of CPU time for a given solution accuracy.

Another interesting coupling of the convergence progress to the discretization concerned the grid density. For a given discretization order, the same "cold"-started Re = 100 problem was run at *globally uniform* resolutions of $h^{-1} = 10$, $h^{-1} = 20$, and $h^{-1} = 40$. The largest of these problems required 90,642 degrees of freedom for its representation. For our cold start, the first Newton step is based on a flow field containing no vorticity singularities and is discussed as a special case immediately below. Immediately following Newton steps required substantially more GMRES steps than the first one at all grid densities. However, the effect was more pronounced at the higher grid densities. Thus, the $h^{-1} = 10$ case jumped from 8 to 31 GMRES iterations between Newton iterations 1 and 2, the $h^{-1} = 20$ case from 11 to 51 GMRES iterations, and the $h^{-1} = 40$ case from 14 to 79 iterations. Since it is not practical to store Krylov subspaces of such high degree for such large problems, we were forced to use restarted GMRES in these tests, which requires more iterations than a full GMRES. We used a maximum Krylov dimension of 40. Newton steps subsequent to the second generally required successively fewer GMRES steps, tapering to fairly small numbers in the last outer iteration. A practical implication from this study is that highly resolved flow computations should be approached through a sequence of grids ranging from coarse to fine, so that much of the numerical shock of vorticity singularities can be distributed at coarser scales and subsequently refined. This practice is, of course, fundamental to the FMV form of multigrid and can be recommended on theoretical and practical grounds in the context of the solution of BVPs by Newton's method; see, e.g., [26].

Finally, we note in the preceding paragraph the logarithmic growth in h^{-1} of the number of GMRES iterations required in the first Newton step. Each doubling of

Table 1. Total execution time, T (in sec), and relative speedup, s, over a range of numbers of processors, p, of the Intel iPSC/860 for five different discretizations of the backstep flow problem at Reynolds number 100, solved by using Newton's method. All data are for a fixed tessellation of 112 tiles. Labels "Global" and "Local" refer to the span of the refined regions, N is the total number of unknowns in the discrete system, and I_1 is the number of GMRES iterations required in executing the first Newton step. Missing entries could not be computed because of memory limitations in smaller clusters of processors. Perfect relative speedups between successive rows would be 2.

	$h_{eff}^{-1} = 10$		$h_{eff}^{-1} = 20$				$h_{eff}^{-1} = 40$		
	Global		Local		Global		Local		Global
	$N = 5,862$		$N = 10,422$		$N = 22,922$		$N = 40,742$		$N = 90,642$
	$I_1 = 8$		$I_1 = 11$		$I_1 = 11$		$I_1 = 13$		$I_1 = 14$
p	T	s	T	s	T	s	T	s	T
2	16.3	-	-	-	-	-	-	-	-
4	10.2	*1.60*	17.2	-	-	-	-	-	-
8	6.3	*1.62*	10.1	*1.70*	20.6	-	-	-	-
16	4.3	*1.47*	6.4	*1.58*	11.7	*1.76*	29.9	-	-
32	3.7	*1.16*	4.7	*1.36*	8.2	*1.42*	17.5	*1.70*	49.3

the mesh density h^{-1} (with the same underlying tessellation) resulted in a constant increase of 3 in the number of iterations required. This follows the theory for the scalar equation summarized in Section 2.1.

With the exception of the discussion of linear problems in the last paragraph (addressed in a larger context in [14]), these remarks must be regarded as specific to the flow configuration studied. We expect, however, that they provide useful rules of thumb for domain-decomposed iterations for nonlinear elliptic BVPs, and we plan to ascertain their generality in a variety of cold and reacting flow configurations in subsequent reports.

4.3. Parallel Performance. We conclude this section with Table 1 showing performance curves for the tile algorithm on the Intel iPSC/860. Because parallel efficiency is crucially dependent upon arithmetic task and processor granularity and load balance, we investigate power-of-two sequences of problem and processor array sizes. Because typical problems are too large to fit on a single processor, we cannot report overall speedups, but report relative speedups with each doubling of processor array size.

Traversing columns, we observe the typical degradation in speedup as processors are added at a fixed problem size. Traversing rows, we observe the typical improvements in speedups as problem size is increased at a fixed processor force. Going down the main diagonal we note that parallel performance is maintained when processor and problem sizes are scaled in proportion. (An exception occurs in the last row ($p = 32$), where systematic load imbalances occur because p does not evenly divide the number of tiles for the first time in the table; thus, half of the nodes have three tiles and the other half have four.) However, we note that overall execution time is not likely to be optimized by indefinite increases in the effective h^{-1} at fixed tessellation; a more complete study would include several (p, h^{-1})-planes like Table 1 at different H^{-1}. The subdomain factorization complexities currently contain terms cubic in h^{-1} and, similarly, the crosspoint factorization complexities contain terms cubic in H^{-1}. These leading terms should be balanced against one another, or the modules contributing them should be replaced with, for instance, multigrid solves. Multigrid makes a particularly attractive solver for large subdomain problems, since the subdomains generally possess greater uniformity than the problem as a whole.

The table also affords a crude indication of the value of adaptive refinement. Comparing the "Local" and "Global" columns at the same h_{eff}^{-1}, we see memory and execution time savings of factors of two or more for local refinement, with the memory savings allowing a smaller feasible number of processors to solve the problem to the same h_{eff}^{-1}.

Though of dubious value in evaluating algorithms, raw performance data on the iPSC/860 may also be of interest. Our aggregate flop rate on 32 processors ranged from about 1.5Mflops in the crosspoint solve phase to 126Mflops in the parallelized matrix-vector multiplies with the operator A for the largest problem of over 90,000 unknowns. For this largest problem, 110Mflops and 99Mflops, respectively, were realized in doing the concurrent subdomain factorizations and backsolves constituting the A_I^{-1} phase of the preconditioner application. Extrapolation of some of these aggregate rates to larger clusters of processors and problem sizes is nontrivial because of both external communication and internal memory hierarchies, but we would expect execution rates for operations like the subdomain factorizations and backsolves to extrapolate roughly linearly in the number of processors, for the same discrete-size tiles.

FORTRAN77 and C compilers for the iPSC/860 are regarded as immature at present. We used the Greenhills compilers with optimizations `-OLM -Z618`. We compared FORTRAN77 and C versions of the most compute-intensive kernels on a model 10×10 tile and selected the fastest of each, which was usually the C version. (The parallel skeleton of the code is entirely in C, but some modules executing sequentially within a processor are in FORTRAN77.) We also tried the Portland Group compiler on our kernels and did not find it to be significantly better at the highest safe optimization level. We believe that there is little room for additional optimization of the arithmetic processing rates relative to supplied hardware and software technology and, therefore, that the speedups do not suffer from any artificial inflation. Because we preserve local uniformity of the data structures, it should be possible to get higher performance from some kernels by making better use of the processor memory caches. The software currently available on the systems to which we have access does not exploit this structure. We prefer to wait for compiler improvements rather than rewrite these kernels in i860 assembly language.

We hope to benefit in the future from better support for global communication along with improved compilers. The GMRES solver relies heavily on global inner products (there are thousands of inner products in a typical execution), so improvements to this one communication-intensive operation will substantially improve the overall parallel efficiency of our code on typical elliptic systems. It is possible to group the inner products within a single GMRES orthogonalization phase in order to make the number of calls to the global reduction routine proportional to the iteration count, rather than to its square in the naive implementation. This optimization has so far been implemented only for the case in which A and B are based on the same discretization.

With an eye towards applications, we note that in the present code approximately 97% of the execution time is consumed in the linear algebra modules. This includes 83% of the time in the preconditioner, 5% of the time in the matrix-vector multiplies, and 9% of the time in GMRES apart from calls to form the action of A and B^{-1}. The preconditioner work breaks down, in turn, into 59% of the total time in backsolves and 24% in factorizations. The evaluation of the coefficients of the operators A and B and the computation of the nonlinear residuals of the streamfunction-vorticity system accounts for only about 3% of the total execution time. In our experience with solving reacting flow problems with detailed models for the chemical kinetics and transport on serial computers, the nonlinear residual and Jacobian evaluation phases of the calculations can themselves consume the dominant share of execution time. As models with more complex source terms and multicomponent transport laws are added to the present code, we expect improved parallel efficiencies, since ratio of local operations to neighbor data exchanges is higher in such problems.

5. Concluding Remarks.

As demonstrated by adaptively refined parallel computations of nonlinear, non-selfadjoint, multicomponent model fluid flow problems, domain decomposition is maturing as a practical algorithmic paradigm for engineering applications. Among various types of divide-and-conquer algorithms, two-scale preconditioned domain-decomposition is a natural compromise between the requirements of the problem physics, current parallel hardware, and maintainable, portable software. However, much research remains to be performed before previously inaccessible computations, such as complex multidimensional convection-diffusion-reaction systems, become quotidian.

Theoretically, more guidance in the construction of general-purpose preconditioners is needed. Known optimal three-dimensional preconditioners for nonoverlapping decompositions are very cumbersome to program. In two dimensions further research is needed on interface preconditioners for multicomponent problems and on multilevel preconditioners, to remove the burden of a "too fine" coarse-grid solve.

From a parallel computing perspective, the main unresolved issue in domain decomposition is the trade-off between good load balance and good data locality. This is common to many problems in parallel computation. An issue to be addressed in the future is mapping onto massively parallel computers consisting of MIMD clusters of SIMD arrays. The two-level tile algorithm seems ideally suited to such an architecture, as discussed briefly in [14].

Advances in automatic adaptive discretization techniques from the past decade [12] need to be incorporated into domain decomposition software. Building libraries of tiles is one convenient way to aid this effort in the context of the current algorithm.

Finally, as with any powerful solution algorithm, preconditioned domain decomposition iterative techniques need to be integrated into complete supercomputing environments in order to make testing on genuine engineering applications convenient. User-interactive problem definition, visualization, and computational steering (particularly of nonlinear problems) are needed. By relying less on global structure than many solution algorithms and providing much in the way of local structure to powerful nodes, domain decomposition is a natural algorithmic bridge between applications and architectures.

Acknowledgments. We are grateful for the opportunity to run on the Intel iPSC/860 hypercube at the Institute for Computer Applications in Science and Engineering, NASA-Langley Research Center. We thank Thomas W. Crockett for his management of the resource, and Director Robert G. Voigt for his partial sponsorship of this research.

REFERENCES

[1] M. J. Berger and S. H. Bokhari, A Partitioning Strategy for Non-uniform Problems across Multiprocessors, IEEE Trans. on Comput., C-36 (1987), pp. 570-580.

[2] J. H. Bramble, J. E. Pasciak, and A. H. Schatz, The Construction of Preconditioners for Elliptic Problems by Substructuring, I, Math. Comp., 47 (1986), pp. 103-134.

[3] A. Brandt, The Weizmann Institute Research in Multilevel Computation: 1988 Report, in *Proceedings of the Fourth Copper Mountain Conference on Multigrid Methods*, J. Mandel, S. F. McCormick, J. E. Dendy, Jr., C. Farhat, G. Lonsdale, S. V. Parter, J. W. Ruge, and K. Stüben, eds., SIAM, Philadelphia, 1989, pp. 13-53.

[4] X.-C. Cai, An Additive Schwarz Algorithm for Nonselfadjoint Elliptic Equations, in *Third International Symposium on Domain Decomposition Methods for Partial Differential Equations*, T. F. Chan, R. Glowinski, J. Périaux, and O. B. Widlund, eds., SIAM, Philadelphia, 1990, pp. 232-244.

[5] X.-C. Cai, W. D. Gropp, and D. E. Keyes, Convergence Rate Estimate for a Domain Decomposition Method, Yale Univ., Dept. of Comp. Sci., RR-827, October 1990.

[6] T. F. Chan and D. E. Keyes, Interface Preconditionings for Domain-Decomposed Convection-Diffusion Operators, in *Third International Symposium on Domain Decomposition Methods for Partial Differential Equations*, T. F. Chan, R. Glowinski, J. Périaux, and O. B. Widlund, eds., SIAM, Philadelphia, 1990, pp. 245-262.

[7] M. K. Denham and M. A. Patrick, Laminar Flow over a Downstream-facing Step in a Two-dimensional Flow Channel, Trans. Inst. Chem. Engrs. 52 (1974), pp. 361-367.

[8] M. Dryja and O. B. Widlund, An Additive Variant of the Schwarz Alternating Method for the Case of Many Subregions, NYU, Courant Institute TR 339, December 1987.

[9] S. C. Eisenstat, H. C. Elman, and M. H. Schultz, Variational Iterative Methods for Nonsymmetric System of Linear Equations, SIAM J. Numer. Anal. 20 (1983), pp. 345-357.

[10] H. C. Elman, Y. Saad, and P. E. Saylor, A Hybrid Chebyshev-Krylov Subspace Algorithm for Solving Nonsymmetric Systems of Linear Equations, Yale Univ., Dept. of Comp. Sci., RR-301, February 1984.

[11] P. F. Fischer and A. T. Patera, Parallel Spectral Element Methods for the Incompressible Navier-Stokes Equations, in *Solution of Superlarge Problems in Computational Mechanics*, J. H. Kane, A. D. Carlson and D. L. Cox, eds., Plenum, New York, 1989, pp. 49-65.

[12] J. E. Flaherty, P. J. Paslow, M. S. Shephard, and J. D. Vasilakis, eds., *Adaptive Methods for Partial Differential Equations*, SIAM, Philadelphia, 1989.

[13] W. D. Gropp and D. E. Keyes, Domain Decomposition on Parallel Computers, Impact of Comput. in Sci. and Eng. 1 (1989), pp. 421-439.

[14] W. D. Gropp and D. E. Keyes, Domain Decomposition with Local Mesh Refinement, Inst. for Comp. Appl. in Sci. and Eng., Technical Report 91-19, February 1991.

[15] W. D. Gropp and D. E. Keyes, Parallel Performance of Domain-Decomposed Preconditioned Krylov Methods for PDEs with Adaptive Refinement, Yale Univ., Dept. of Comp. Sci., RR-773, March 1990.

[16] W. D. Gropp and D. E. Keyes, Parallel Domain Decomposition and the Solution of Nonlinear Systems of Equations, Mathematics and Computer Science Preprint MCS-P186-1090, Argonne National Laboratory, October 1990.

[17] D. E. Keyes and W. D. Gropp, Domain-Decomposable Preconditioners for Second-Order Upwind Discretizations of Multicomponent Systems, Mathematics and Computer Science Preprint MCS-P187-1090, Argonne National Laboratory, October 1990.

[18] J. M. Leone, Jr., and P. M. Gresho, Finite Element Simulations of Steady, Two-Dimensional, Viscous Incompressible Flow over a Step, J. Comp. Phys. 41 (1981), pp. 167-191.

[19] N. M. Nachtigal, L. Reichel, and L. N. Trefethen, A Hybrid GMRES Algorithm for Nonsymmetric Linear Systems, Proceedings of the Copper Mountain Conference on Iterative Methods, April 1990.

[20] D. M. Nicol, J. H. Saltz, and J. C. Townsend, Delay Point Schedules for Irregular Parallel Computations, Int. J. Parallel Processing 18 (1989), pp. 69-90.

[21] A. T. Patera, A Spectral Element Method for Fluid Dynamics: Laminar Flow in a Channel Expansion, J. Comp. Phys. 48 (1984), pp. 468-488.

[22] P. J. Roache, *Computational Fluid Dynamics*, Hermosa, Albuquerque, 1972.

[23] Y. Saad and M. H. Schultz, GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems, SIAM J. Sci. Stat. Comp. 7 (1986), pp. 865-869.

[24] P. E. Saylor and D. C. Smolarski, Implementation of an Adaptive Algorithm for Richardson's Method, Eidgenössische Technische Hochschule Zürich, Dept. of Informatics, Inst. for Scientific Computing, Report 139, October 1990.

[25] M. D. Smooke, Solution of Burner-Stabilized Pre-Mixed Laminar Flames by Boundary Value Methods, J. Comp. Phys. 48 (1982), pp. 72-105.

[26] M. D. Smooke and R. M. M. Mattheij, On the Solution of Nonlinear Two-Point Boundary Value Problems on Successively Refined Grids, Appl. Num. Math. 1 (1985), pp. 463-487.

[27] H. A. Van der Vorst, Bi-CGSTAB: A More Smoothly Converging Variant of CG-S for the Solution of Nonsymmetric Linear Systems, July 1990. (Manuscript)

[28] O. C. Zienkiewiczi, J. Z. Zhu, A. W. Craig, and M. Ainsworth, Simple and Practical Error Estimation and Adaptivity: h and $h-p$ Version Procedures, in *Adaptive Methods for Partial Differential Equations*, J. E. Flaherty, P. J. Paslow, M. S. Shephard, and J. D. Vasilakis, eds., SIAM, Philadelphia, 1989, pp. 100-114.

Report Documentation Page

National Aeronautics and Space Administration

1. Report No. NASA CR-187529 ICASE Report No. 91-20	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle DOMAIN DECOMPOSITION METHODS IN COMPUTATIONAL FLUID DYNAMICS		5. Report Date February 1991
		6. Performing Organization Code
7. Author(s) William D. Gropp David E. Keyes		8. Performing Organization Report No. 91-20
		10. Work Unit No. 505-90-52-01
9. Performing Organization Name and Address Institute for Computer Applications in Science and Engineering Mail Stop 132C, NASA Langley Research Center Hampton, VA 23665-5225		11. Contract or Grant No. NAS1-18605
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Langley Research Center Hampton, VA 23665-5225		13. Type of Report and Period Covered Contractor Report
		14. Sponsoring Agency Code

15. Supplementary Notes

Langley Technical Monitor:
Michael F. Card

Submitted to International Journal of Numerical Methods in Fluids

Final Report

16. Abstract

The divide-and-conquer paradigm of iterative domain decomposition, or substructuring, has become a practical tool in computational fluid dynamics applications because of its flexibility in accommodating adaptive refinement through locally uniform (or quasi-uniform) grids, its ability to exploit multiple discretizations of the operator equations, and the modular pathway it provides towards parallelism. We illustrate these features on the classic model problem of flow over a backstep using Newton's method as the nonlinear iteration. Multiple discretizations (second-order in the operator and first-order in the preconditioner) and locally uniform mesh refinement pay dividends separately, and they can be combined synergistically. We include sample performance results from an Intel iPSC/860 hypercube implementation.

17. Key Words (Suggested by Author(s)) domain decomposition, preconditioning, Kyrlov methods, Newton's method, computational fluid dynamics, parallel computing	18. Distribution Statement 34 - Fluid Mechanics and Heat Transfer 64 - Numerical Analysis Unclassified - Unlimited

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of pages 24	22. Price A03