Contemporary Mathematics
Volume 157, 1994

Multigrid Solvers on Decomposed Domains

ACHI BRANDT and BORIS DISKIN

ABSTRACT. For general nonlinear elliptic problems with many gridpoints per processor, a domain-decomposed multigrid algorithm is described. It solves a problem in essentially the same work as needed for solving just once, by the fastest solver, a separate problem in each subdomain. During the entire solution process, only few episodes of data transfer between processors are needed, and the total amount of transferred data is small compared with the size of the decomposition interfaces. A mode analysis and numerical tests are reported.

1. Introduction

The fastest solvers for discretized elliptic partial differential equations on scalar computers are the full-multigrid (FMG) algorithms. Such an algorithm, described below, can, for example, solve the 5-point Poisson equation in a general domain to errors below the discretization errors in about 22 computer operations per gridpoint; each additional multigrid cycle, costing about 15 operations per gridpoint, reduces the errors by more than another order of magnitude. This same efficiency, with the number of operations just growing proportionately to the number of operations involved in describing the discretized system, has been obtained for general nonlinear elliptic systems in general domains [4], including elasticity and flow problems, and including systems with strongly discontinuous coefficients [1], or with strongly twisted topologies (Dirac equations [7]). Recently, similar efficiency has been obtained for non-elliptic problems, including high-Reynolds flow problems [10], [11], and also for highly indefinite systems.

1991 Mathematics Subject Classification. Primary 65N55, 65N22.
Supported in parts by grants AFOSR-91-0156 from the United States Air Force, and NSF DMS-9015259 from the American National Science Foundation.
The final version of this paper will be submitted for publication elsewhere.

Moreover, even for $n \times n$ *dense*-matrix problems arising in science and engineering (from integral and integro-differential equations, or from many particle interactions) multigrid solvers still typically require only $O(n)$ or $O\big(n(\log n)^q\big)$ operations [6], [8].

The first purpose of the present study is to demonstrate that this same efficiency — i.e., nearly the same total number of operations per gridpoints — can be obtained when the problem is solved by any number, P, of processors working in parallel, provided there are many gridpoints per processor ("coarse granularity"). This means that the solution time is reduced by the factor $1/P$.

Moreover, the purpose is also to address the situation where communication between the processors cannot be too frequent and is relatively expensive. For example, the set of parallel processors may in fact be a cluster of workstations communicating via a local network. It will be demonstrated that the number of inter-processor data-transfer *episodes* can be limited to only $O(\log n)$, where n is the total number of grid points. Furthermore, for elliptic problems it will be demonstrated that the amount of data transferred at each such episode can be surprisingly small.

A common approach to the low-communication coarse-granularity situation is to decompose the domain of the problem into P subdomains, and to assign one processor to solve the equations inside each subdomain, leaving some equations unsolved and some values unchanged on the inter-subdomain interfaces. An adjustment is then made to the interfaces, and the equations at each subdomain are resolved with the new interface values. This is repeated iteratively until convergence is obtained. Each solution process within each subdomain can of course employ a multigrid solver.

The approach here will be quite different. The multigrid algorithm will be employed only once, to the entire system of equations on the whole domain. Each step of the algorithm will be performed in parallel by the P processors, each working in its own subdomain. By creating a small overlap between the subdomains on each of the multigrid levels, it will be shown that inter-processor data transfers can be made only once per multigrid cycle, each processor transferring data only to neighboring processors. Moreover, it will be shown that the total number of values that need be transferred from each processor is much smaller than the number of gridpoints on its interfaces.

The total arithmetic work of such algorithms is essentially equivalent to solving just *once*, and by the *fastest* (the FMG) solver, in each subdomain. The algorithms themselves are simple modifications of the ordinary multigrid algorithms. In their FAS version presented below, they are directly applicable to *nonlinear* problems: no linearization is needed and the efficiency is the same as for linear problems.

2. Differential and discretized problems

We will consider general nonlinear elliptic problems in an open domain Ω with boundary $\partial\Omega$. The differential equation

$$Lu(x) = f(x) \qquad (x \in \Omega) \tag{2.1}$$

is given, where f is a known function, u is the unknown one and L is a uniformly elliptic operator. The algorithm described below will be suitable for the general

case that (2.1) is a system of q nonlinear differential equations in q unknown functions $u(x) = \left(u^1(x), \ldots, u^q(x)\right)^T$ (see details in [4]); but for convenience the reader can refer to the scalar ($q = 1$) case, or, more specifically, to our standard example, the two dimensional Poisson equation: $x = (x_1, x_2)$, $L = \partial^2/\partial x_1^2 + \partial^2/\partial x_2^2$. While the outline will be general, full specification of the algorithm will be given in terms of this particular example. Only in Sec. 5.5 we will mention another, in fact nonlinear, example.

In addition to the interior equation (2.1), suitable boundary conditions will be assumed on $\partial\Omega$. In our standard example this will be the Dirichlet boundary condition

$$u(x) = g(x) \qquad (x \in \partial\Omega). \tag{2.2}$$

The discretization for the algorithms considered here can be fairly general, involving finite element or finite differences on non-uniform, possibly staggered grids of various types. Again for simplicity, our examples will be in terms of finite difference equations on uniform grids. For any grid Ω_h with meshsize h covering the domain Ω, the discretization of (2.1) will be written

$$L^h u_i^h = f_i^h \qquad (i \in \Omega_h) \tag{2.3}$$

where $i = x/h$ is an integer vector. In the standard example $i = (i_1, i_2) = (x_1/h, x_2/h)$, where i_1 and i_2 are integers, and L^h is the five-point Laplacian. Also for simplicity, we treat here only problems where $\partial\Omega$ coincides with the gridlines of all our grids, in which case the implementation of the boundary condition (2.2) is straightforward.

3. Regular FAS-FMG solver

For the fast solution of (2.3), a sequence of grids $\Omega^1, \Omega^2, \ldots, \Omega^M$ is employed, where $\Omega^k = \Omega_{h_k}$, $h_k = h_{k-1}/2$, $h_M = h$, and where Ω^1 contains only few points. Note that gridpoint i in Ω^{k-1} has the same physical position x as gridpoint $2i$ in Ω^k. On each grid Ω^k, a discretization analogous to (2.3) is given

$$L^k u_i^k = f_i^k \qquad (i \in \Omega^k) \tag{3.1}$$

where the notation is switched to $L^k = L^{h_k}$, $u^k = u^{h_k}$, $f^k = f^{h_k}$.

For $k = 1$ the system (3.1) is so small that a very fast solver for it is easy to construct "directly" (meaning either a genuine direct solver or fast converging iterations). For $k > 1$, multigrid solvers are described below. The approximate solution at any stage will be denoted $\tilde{u}^k$; the solution process will thus be described as a sequence of improvements for $\tilde{u}^k$.

3.1. The FAS cycle $FAS_k(\nu_1, \nu_2; \gamma)$. Given an equation on level $k > 1$ of the form (3.1), and an initial approximate solution $\tilde{u}^k$, *the Full Approximation Scheme (FAS) cycle* $FAS_k(\nu_1, \nu_2; \gamma)$ *for improving this approximation* is recursively defined as the following seven steps.

(i) *Pre-relaxation.* Improve $\tilde{u}^k$ by ν_1 relaxation sweeps. In our standard example, we will employ Gauss-Seidel relaxation in red-black ordering: in the

first half of each sweep all the "red" equations (equations at points (i_1, i_2) where i_1+i_2 is even) are relaxed, and in the second half — the rest of them (the "black" ones, with $i_1 + i_2$ odd).

(ii) *Solution transfer.* Form an approximation to the current solution $\tilde{u}^k$ on the next coarser grid Ω^{k-1}. We will denote this approximation $\hat{I}_k^{k-1}\tilde{u}^k$. In our standard example, $\hat{I}_k^{k-1}$ will be simple "*injection*", i.e.,

$$(\hat{I}_k^{k-1}\tilde{u}^k)_{i_1,i_2} = \tilde{u}^k_{2i_1,2i_2}. \tag{3.2}$$

(iii) *Residual transfer.* Also form on the next coarser grid an approximation to the current residual function $r^k = f^k - L^k\tilde{u}^k$. This coarse grid approximation will be denoted $I_k^{k-1}r^k$. In our standard example, I_k^{k-1} will be the usual "*full weighting*" operator, i.e.,

$$\begin{aligned}(I_k^{k-1}r^k)_{i_1,i_2} &= \tfrac{1}{4}r^k_{2i_1,2i_2}\\ &+ \tfrac{1}{8}(r^k_{2i_1+1,2i_2} + r^k_{2i_1-1,2i_2} + r^k_{2i_1,2i_2+1} + r^k_{2i_1,2i_2-1})\\ &+ \tfrac{1}{16}(r^k_{2i_1+1,2i_2+1} + r^k_{2i_1+1,2i_2-1} + r^k_{2i_1-1,2i_2+1} + r^k_{2i_1-1,2i_2-1}).\end{aligned} \tag{3.3}$$

(iv) *Coarse grid equation.* Form the coarser grid equation

$$L^{k-1}u^{k-1} = \tilde{f}^{k-1} \tag{3.4}$$

by calculating

$$\tilde{f}^{k-1} = L^{k-1}(\hat{I}_k^{k-1}\tilde{u}^k) + I_k^{k-1}r^k. \tag{3.5}$$

(v) *Recursion.* If $k-1 = 1$, solve (3.4) directly. Otherwise, starting from $\hat{I}_k^{k-1}u^k$ as the first approximation, improve the solution to (3.4) by γ cycles $FAS_{k-1}(\nu_1, \nu_2; \gamma)$. In either case, denote the final approximate solution to (3.4) by $\tilde{u}^{k-1}$.

(vi) *Coarse grid correction.* Correct the (fine) grid approximate solution by

$$\tilde{u}^k \leftarrow \tilde{u}^k + I_{k-1}^k(\tilde{u}^{k-1} - \hat{I}_k^{k-1}\tilde{u}^k), \tag{3.6}$$

where I_{k-1}^k denotes an interpolation from Ω^{k-1} to Ω^k, which is called *the correction interpolation*. In our examples, I_{k-1}^k is the bilinear interpolation. (General rules governing both I_k^{k-1} and I_{k-1}^k for general systems are given in [4, §4.3], and in more detail in [5].)

(vii) *Post-relaxation.* Finally, improve $\tilde{u}^k$ by ν_2 additional relaxation sweeps, similar to those in Step (i).

This description of the multigrid FAS cycle neglects to mention the extra processing needed at each step in and near the boundaries. Indeed, in the simple example treated here, no such special processing is needed. For the multigrid treatment of general boundaries and general boundary conditions, see [4], [5], [9] and [12].

Observe that in the linear case the correction function $v^{k-1} = u^{k-1} - \widehat{I}_k^{k-1}\widetilde{u}^h$ satisfies the equation $L^{k-1}v^{k-1} = I_k^{k-1}r^k$, approximating the fine-grid residual equation $L^k(u^k - \widetilde{u}^k) = r^k$. Furthermore, even in the nonlinear case, it is easy to see from (3.4)–(3.5) that $I_k^{k-1}r^k$ similarly drives u^{k-1} to differ from $\widehat{I}_k^{k-1}\widetilde{u}^k$. Indeed, the FAS cycle has been shown to be as efficient for many nonlinear problems as for linear ones. In linear cases, v^{k-1} is independent of $\widehat{I}_k^{k-1}\widetilde{u}^k$, so one could for example take $\widehat{I}_k^{k-1}\widetilde{u}^k = 0$. This particular scheme is called the Correction Scheme (CS), since its coarse-grid solution u^{k-1} coincides with the correction function v^{k-1}.

The parameter γ (the number of next-coarser-grid cycles per fine grid cycle) is called *the cycle index*. Cycles with $\gamma = 1$ are called V cycles and denoted $V(\nu_1, \nu_2)$; those with $\gamma = 2$ are termed W cycles and denoted $W(\nu_1, \nu_2)$. For uniformly elliptic problems either of these is used, usually with $\nu_1 + \nu_2 = 2$ or 3. Each such cycle typically reduces the error at least by an order of magnitude, provided a proper treatment is employed at and near the boundaries (see [5], [12]). In our standard example, each of the cycles $V(0,2)$, $V(1,1)$ and $V(2,0)$ with proper boundary treatment reduces the error (in some appropriate norm) by a factor close to .063. With many levels and without boundary treatment, experiments show that the factor rises to .12 for $V(1,1)$ and $V(0,3)$, and to .18 for $V(0,2)$.

3.2. The full multigrid algorithm $FMG_k(\nu_0, \nu_1, \nu_2; \gamma; n)$. In the Full-Multi-Grid (FMG) algorithm, an approximation to the solution of (3.1) is first obtained from the next coarser grid, and then it is improved by relaxation sweeps and multigrid cycles such as $FAS_k(\nu_1, \nu_2; \gamma)$. Thus, *the algorithm* $FMG_k(\nu_0, \nu_1, \nu_2; \gamma; n)$ *for solving* (3.1) (with $k \geq 2$) is recursively defined as the following four steps.

(I) *Coarse grid equation.* The equation

$$L^{k-1}u_i^{k-1} = f_i^{k-1} \qquad (i \in \Omega^{k-1}) \tag{3.7}$$

is formed by calculating

$$f^{k-1} = I_k^{k-1}f^k, \tag{3.8}$$

where I_k^{k-1} is a suitable fine-to-coarse transfer; the full-weighting operator, defined by (3.3), has been used by us for this step as well. Note that in (2.3) and (3.1) we have not specified how f^k is calculated at the various levels ($k = 1, 2, \ldots, M$). It could be obtained at each level directly from f, the differential-equation right-hand side, but in the FMG algorithm each f^{k-1} is obtained by properly averaging f^k, ($k = M, M-1, \ldots, 3, 2$); only f^M, the finest-grid right-hand side, is calculated from f. This guarantees the proper performance of the algorithm for *any* (not just smooth) f. (See the discussion at the end of Sec. 5.4.) The discretization of the boundary conditions could be handled similarly: in case of (2.2), the values g^{k-1} of the boundary conditions on $\partial\Omega^{k-1}$ could be obtained by fully averaging g^k: This however is less important.

(II) *Recursion.* If $k-1 = 1$, solve (3.7) directly. Otherwise, apply algorithm $FMG_{k-1}(\nu_0, \nu_1, \nu_2; \gamma; n)$ for solving (3.7). Denote the obtained approximate solution $\widetilde{u}^{k-1}$.

(III) *Solution interpolation.* Interpolate $\widetilde{u}^{k-1}$ to Ω^k, denoting the result $\Pi_{k-1}^k \widetilde{u}^{k-1}$. Generally this "solution interpolation" Π_{k-1}^k (also called "the FMG interpolation") has higher order than the "correction interpolation" I_{k-1}^k used in (3.6) above; see general rules in [4, §7.1]. In our standard example, Π_{k-1}^k is bicubic.

(IV) *Improvement relaxation and cycles.* Starting with $\widetilde{u}^k = \Pi_{k-1}^k \widetilde{u}^{k-1}$ as the first approximation, improve the solution to (3.1) first by ν_0 initial relaxation sweeps, and then by n cycles of the type $FAS_k(\nu_1, \nu_2; \gamma)$.

The grid Ω^k (or the level k) in this description of the FMG algorithm is called the *currently* finest grid (or level). By contrast, Ω^M (or M), the finest grid (level) for which the algorithm is to be applied, is called *the target* finest grid (level).

For uniformly elliptic problems, a cheap algorithm such as $FMG_k(0,1,1;1;1)$ or $FMG_k(0,2,1;1;1)$ is easily enough to obtain a solution error $\|\widetilde{u}^k - u^k\|$ substantially smaller than the discretization error $\|u^k - I^k u\|$, where $I^k u$ is a representation of the solution of (3.1) on Ω^k and $\|\cdot\|$ is a suitable norm; see examples below.

4. Trivial domain decomposition

The multigrid cycles and algorithms are directly amenable for massive parallel processing. Each step described above can be done in parallel at all gridpoints, or at least, as in the case of the red-black Gauss-Seidel relaxation, at all gridpoints of the same "color" (see [3]). How to divide this processing between actual parallel processors, how much and how often inter-processor information should be transferred, etc., depend on the situation. In this article we discuss the situation typical to domain decomposition algorithms. Namely, we assume that the number of gridpoints in the target finest grid Ω^M is much larger than the number of processors P, so that each processor p can be assigned its own subgrid of operation Ω_p^M, where $\Omega^M = U_{p=1}^P \Omega_p^M$. For convenience of description we will assume that Ω_p^M and Ω_q^M are disjoint for $q \neq p$, so each processor would in principle need information from neighboring subgrids to perform its operations.

The division of Ω^M into subgrids naturally induces a similar division at all coarser levels:

$$\Omega^k = U_{p=1}^P \Omega_p^k; \qquad (k = 1, \ldots, M) \tag{4.1}$$

e.g., in our standard example, $i \in \Omega_p^{k-1}$ iff $2i \in \Omega_p^k$. Note that, on some of the coarsest levels k, some of the subgrids Ω_p^k may well be empty.

A trivial conversion of the above FAS-FMG solver into an algorithm using the P processors in parallel is simply to have each processor p execute all those operations whose results change values at gridpoints belonging to Ω_p^k. Two general problems arise with this algorithm.

First, since some Ω_p^k are empty, some processors will remain idle part of the time. In this paper we assume that there are many gridpoints in each Ω_p^M, so that most of the computing time is spent at the finest levels. Hence we will

ignore here any such very-coarse-level waste. This is especially justifiable since, dealing here only with uniformly elliptic problems, we can confine ourselves to V cycles.

The second problem, which we cannot ignore, is the problem of communication — data transfer and synchronization — between the processors. For simplicity we will assume in the discussion below that each processor has its own memory, not shared by others. Values in one subgrid needed for the processing in another subgrid will be called *interface* values.

A trivial, but perhaps not always satisfactory, solution to the communication problem is to transfer information at particular "*transfer stages*", with a proper definition of such stages. Thus, in our standard example, the transfer stages of the FAS cycle can be defined as follows: (1) before each half (red or black) relaxation sweep, transferring interface values for the appropriate color; (2) before residual transfer, transferring the information needed for calculating both (3.3) and (3.5); (3) before the coarse grid correction (3.6). Similar transfer stages are added before Step I and before Step II of the FMG algorithm.

This simple implementation of the FAS-FMG solver may already be very efficient compared with domain decomposition algorithms. The total work is equivalent to that of solving (by the fastest solver!) only *once* in each subdomain. The amount of information transferred at each stage is at most proportional to the size of the inter-subdomain interfaces; in fact, at most stages (the coarser level stages) the amount is much less. And, by employing V cycles, the number of transfer stages is only $O\big((\log n_M)^2\big)$, where n_M is the number of gridpoints in the target finest level M.

Furthermore, it is immediately clear that both the amount of transferred data and the number of transfer stages can be radically reduced by various algorithmic modifications, such as the following three.

1. Transfers at coarse levels can be reduced by "unifying" subdomains, i.e., assigning one processor to several subgrids, transfering to it all the information from all the processors previously assigned to those subgrids. This should be done, e.g., so as to keep the algebraic work time of each processor comparable to the time spent in the transfer stages. In line with our premise above, such very-coarse-level savings are not studied in this article.

2. Instead of a transfer once per half relaxation sweep, one can transfer once per full sweep, transferring all interface values at once (instead of one color at a time). The result of this is that the relaxation will not be exactly Gauss-Seidel: The relaxation at *some* particular points will not use the *latest* calculated values at *all* their neighbors. This, however, will have only very slight effect, if at all, on the performance of the algorithm.

3. By creating certain overlaps between the subgrids, information can be transferred less often.

Instead of describing all possible alternatives, we will study in the next sections some of the more extreme modifications, whereby transfers can be made only once per cycle, and data corresponding to several of the finest grids need not be transferred at all.

5. Once-per-cycle transfers

5.1. Overlapping subdomains. Similar to the above description, the problem domain Ω will be decomposed into subdomains $\Omega = \bigcup_{p=1}^{P} \Omega_p$, inducing a decomposition at each level $\Omega^k = \bigcup_{p=1}^{P} \Omega_p^k$, where $\Omega_p^k \subseteq \Omega_p$. Here, however, the subdomains Ω_p will be closed, and the border lines between them will be chosen on gridlines of some very coarse level (e.g., the level K discussed in Sec. 5.2, which is effectively the coarsest level of the current algorithm). Hence, the subgrids Ω_p^k will not be quite disjoint: at each level k, gridpoints on border lines will belong to two subgrids, and gridpoints at interior corners will belong to four subgrids.

To each subgrid Ω_p^k we will add a margin of neighboring gridpoints for which solution "ghost values" will be calculated by, and stored at, the p-th processor, although at the same time different values may be assigned to them by other (neighboring) processors. Thus, we define *extended* subgrids $\Omega_p^{k,j}$ as follows: $\Omega_p^{k,0} = \Omega_p^k$, and $\Omega_p^{k,j+1}$ is the set of all gridpoints at distance h_k (or less) from the points of $\Omega_p^{k,j}$. This implies that, for any fixed k and j, neighboring extended subgrids $\Omega_p^{k,j}$ will have an *overlap* of $2j+1$ gridlines: the borderline plus j lines on each of its sides will have double values.

In each of the algorithms below, each processor p will have its own approximate-solution value $\widetilde{u}_i^{k,p}$ and its own right hand side value $\widetilde{f}_i^{k,p}$ defined for each gridpoint $i \in \widehat{\Omega}_p^k = \Omega_p^{k,J(k)}$, where the $J(k)$ are the "*overlap parameters*" of the algorithm. Each gridpoint $i \in \Omega^k$ which is not on border line has a unique processor p_i such that $i \in \Omega_{p_i}^{k,0}$; we define $\widetilde{u}_i^{k,p_i}$ to be the *genuine value* of $\widetilde{u}^k$ at i, and denote it $\widetilde{u}_i^k$. For gridpoint i belonging to a border line the genuine value $\widetilde{u}_i^k$ is defined to be the average of $\widetilde{u}_i^{k,p}$ over all p such that $i \in \Omega_p^{k,0}$.

Genuine right-hand side values $\widetilde{f}_i^k$ will be defined recursively. At the currently finest level $\widetilde{f}_i^k = f_i^k$, and on coarser levels $\widetilde{f}_i^{k-1}$ is defined by (3.5), where $r^k = \widetilde{f}^k - L^k\widetilde{u}^k$ is based on the genuine values of $\widetilde{f}^k$ and $\widetilde{u}^k$.

5.2. The coarsest level. In the algorithms below, data is transferred between processors once per cycle. This is done when the cycle reaches a (very) coarse level K. Typically K will be chosen so that the number of gridpoints at level K is at most comparable to the number of points in the borderlines between subdomains of the (target) finest grid Ω^M. This means that $h_K \geq O(h_M^{(d-1)/d})$, where d is the dimension of our problem.

The interprocessor transfers will be of two types: data exchange in the overlap regions of all the levels $K < k \leq M$, and interactions with one common coarse problem created on Ω^K. Taking the basic work attitude of the present study (to ignore the work on grids much coarser than the finest), we do not investigate the question of how the Ω^K problem is actually solved, whether by one processor into which all information is transferred, or by further collaboration of several processors. The fast solution of the Ω^K problem can again use a multigrid algorithm employing still coarser grids, but since we will not be interested in how that is done, we can regard K as our *coarsest* level, renumbering the

levels so that $K = 1$.

5.3. The domain-decomposed cycle $DDV_k(\nu_2; J)$. To simplify the algorithm, we base it on the V cycle without pre-relaxation, i.e., on the cycle $V_k(0, \nu_2) = FAS_k(0, \nu_2; 1)$. The first "leg" of this cycle sends solution and averaged residual values from the finest level k to the next coarser level $k-1$, then to level $k-2$, and so on to the coarsest level 1, without any intermediate relaxation sweeps, calculating on the way $\widetilde{f}^{k-1}, \widetilde{f}^{k-2}, \ldots, \widetilde{f}^{1}$.

The same can be done in parallel by the P processors, with processor p sending the genuine values of u_i^{k,p_i} from Ω_p^k to Ω_p^{k-1}, then to Ω_p^{k-2}, and so on to Ω_p^1, and also sending residuals and calculating from them genuine values of $\widetilde{f}_i^{\ell}$ at points i in the *interior* of Ω_p^{ℓ} ($\ell = k-1, k-2, \ldots, 1$). Then, at this stage, the inter-processor data transfer takes place, in which *non-genuine values are replaced by genuine values*. That is, the genuine values $\widetilde{u}_i^k$ and borderline solution values $\widetilde{u}_i^{k,p}$ are transferred to all points i in $\widehat{\Omega}_p^k$ carrying non-genuine values, and so are the genuine values of $\widetilde{f}_i^k$ wherever available. The overlap parameters $J(k)$ used in this algorithm are independent of the level: $J(2) = J(3) = \cdots = J(k) = J$. Note that before the transfer, for some gridpoints i on or near the interior boundaries genuine values of $\widetilde{u}_i^k$ and $\widetilde{f}_i^k$ could not be calculated by any of the processors; they are calculated *after* the transfer, using the just-transferred solution values.

After solving the problem at the coarsest level 1, the second "leg" of the V_k cycle normally proceeds to level 2, then to 3, and so on back to level k. At each level, a coarse grid correction (like (3.6)) is first made, followed by ν_2 relaxation sweeps. If $J \geq 4\nu_2$, this process can precisely be implemented by the P processors in parallel, with no further interprocessor transfer. Indeed, when the processor p executes these steps in $\widehat{\Omega}_p^2$, the values it calculates at distance $2\nu_2 h_2$ or more from the interior boundaries of $\widehat{\Omega}_p^2$ will be (in the case of red-black Gauss-Seidel relaxation) genuine values, hence all the values interpolated to $2\widehat{\Omega}_p^3$ will be genuine; and so on to level k.

The same processing can also be done even if $J < 4\nu_2$, except that non-genuine values of $\widetilde{u}_i^{\ell,p}$ will propagate in $\widehat{\Omega}_p^{\ell} - \Omega_p^{\ell}$ until they will affect values at Ω_p^{ℓ}, if not immediately at level $\ell = 2$ then possibly at some finer level ℓ. Thus, in this case, the results of this parallel processing cycle, which will be denoted $DDV_k(\nu_2; J)$, will slightly differ from those of the single processing cycle $V_k(0, \nu_2)$.

The differences will be slight, however, since for h-elliptic equations, the distance to which *significant* effects are propagated by ν_2 relaxation sweeps is much shorter than $2\nu_2$ meshsizes.

Numerical experiments corroborated this. They were conducted with the standard example described above, on an 8×8 square domain, with coarsest-grid meshsize $h_1 = 1$, with $k = 6$ levels, and with $P = 2$ subdomains, Ω_1 and Ω_2, each covering an 8×4 half of the domain. Thus, the finest grid Ω^6 has 257×257 points (including boundary points), and each $\widehat{\Omega}_p^6$ has $257 \times (129 + J)$ points.

The asymptotic convergence factor of the ordinary $V_6(0, 2)$ cycle is .165

per cycle; that of $DDV_6(2;2)$ was found to be roughly the same for a number of cycles, but climbed to .222 after 10 cycles. The results of $DDV_6(2;4)$ were indistinguishable, throughout 10 cycles, from those of $V_6(0,2)$.

These factors are good enough for producing top-efficiency FMG algorithms, which are discussed next.

5.4. The domain-decomposed FMG algorithm $DDF_M(\nu_0,\nu_2;J;n)$. This is a parallel processing algorithm based on the single-processing one $FMG_M(\nu_0,0,\nu_2;1;n)$ described above (Sec. 3.2). The main modification is the use of cycle $DDV_k(\nu_2;J)$ instead of $V_k(0,\nu_2) = FAS_k(0,\nu_2;1)$. Other rather straightforward modifications can be introduced, when needed, to the calculation of the right-hand side (3.8) and to the solution interpolations Π^k_{k-1}: the former can be done similarly to the residual weighting within the DDV_M cycle, including a single stage of inter-processor transfers at the coarsest level; and the latter could use non-central interpolation whenever reliable (e.g., genuine) values are not available on one side of the interpolation point. In practice, the modification of Π^k_{k-1} did not seem to matter much.

Numerical experiments were conducted with our standard example, on the 8×8 domain $\{|x_1|,|x_2|\le 4\}$, decomposed as above to two subdomains. The border line between them is $\{x_1=0\}$, and right-hand sides f and g were chosen so that the solution to (2.1)–(2.2) is $u(x)=\cos\big(A(x_1-4)+B(x_2-4)\big)$. Various values of A and B were tested, to cover any possible kind of components. The algebraic error $\|\widetilde{u}^M-u^M\|$ at various stages of the algorithm $DDF_M(\nu_0;\nu_2;J;n)$ is compared in Table 1 to the discretization error $\|u^M-I^M u\|$, where the norm is the discrete analog of the L_2 norm. The algorithm stages at which the algebraic error is shown are the following: (1) after the solution interpolation $\widetilde{u}^M = \Pi^M_{M-1}u^{M-1}$; (2) after the ν_0 initial relaxation sweeps; (3) after the coarse-grid correction to level M in the first cycle (before relaxing on level M); (4) at the end of the first cycle (after the ν_2 post-relaxation sweeps on level M); (5) after the coarse-grid correction to level M in the second cycle; (6) at the end of the second cycle. (A second cycle on level M can be performed even in the case $n=1$: n shows the number of cycles that have been performed at each *coarser* level of the FMG algorithm.)

The results show that, for any type of solution component, the algorithm $DDF_M(2,2;2;1)$ yields algebraic errors smaller than the discretization errors already at stage (3), i.e., when on the finest level 2 sweeps have been made *before* the cycle, but not after. This algorithm is extremely inexpensive: neglecting interprocessor transfers and overlap repetitions (i.e., assuming sufficiently large grids), the algorithm can be programmed to cost only 24 computer *additions* per level-M gridpoint. (The only multiplications are by integral powers of 2, which in floating-point binary arithmetic can be performed as additions.)

Furthermore, the results show that after an additional cycle (at stage (5) of the table), the algebraic errors are already nearly an order of magnitude smaller than the discretization error. This additional cycle can be programmed to cost only 15 computer additions per level-M gridpoint.

Other features can be noticed in the table, such as: For sufficiently *smooth* u no relaxation at the finest level is needed (see results at stage (3) for the case $A=B=1$, $\nu_0=0$); but the *initial* relaxation sweeps on the finest level

are definitely important for *high-frequency* components: for them, if $\nu_0 = 0$, results are bad, even at stage (4). Modestly smaller (about half as small) errors are obtained by increasing ν_2 from 2 to 3, or increasing n from 1 to 2. For some high-frequency components there appears to be some sensitivity to the exact location of the interface (compare $J = 2$, 4 and 6 in the case $A = 1$, $B = 25$); still, in these as in all other high-frequency cases, the algebraic errors of $DDF_M(2,2;2;1)$ at stage (3) are already *much* smaller than the discretization errors.

Notice also the huge discretization errors for the high-frequency cases $(A, B) = (25, 1)$ or $(1, 25)$ when $M = 3$. This is due to the fact that the procedure for calculating the discretization errors uses a right-hand side f^M which is *injected* from the differential right-hand side f. In these particular cases the injection happens to skip the oscillations in f, exhibiting very smooth f^M instead, thus producing u^M very large compared with u. If one used that solution u^3 as a first approximation to u^4, that initial error would be so large that more than one cycle might be needed to reduce it to the size of $\|u^4 - I^4 u\|$. This is the reason for the weighting (3.8) used in the FMG algorithm.

5.5. Nonlinear example. We have also tried the same algorithm for the nonlinear equation $(1 + u^2)u_{xx} + u_{yy} = f$, which for the cases described above ($|u| \leq 1$) can use the same relaxation scheme: the coefficient $(1+u_i^2)$ can simply be frozen while relaxing the equation at gridpoint i. Results similar to the linear case were obtained. More nonlinear results will be reported in the full version of this paper.

5.6. Reducing the amount of transferred data. Preliminary tests indicate that the above FMG algorithm retains most of its efficiency when much less data is transferred at each level.

In particular, it appears enough to transfer only genuine *solution* values, and only at the interior boundaries; i.e., in the tests described above, only to replace all $\tilde{u}^{k,1}_{J,i_2}$ by $\tilde{u}^{k,2}_{J,i_2}$ and all $\tilde{u}^{k,2}_{-J,i_2}$ by $\tilde{u}^{k,1}_{-J,i_2}$. Within each processor, corrections to other values can then be interpolated from these ones: e.g., for $0 < i_1 \leq J$ and for all i_2, the correction $(i_1/J)(\tilde{u}^{k,2}_{J,i_2} - \tilde{u}^{k,1}_{J,i_2})$ can be added to $\tilde{u}^{k,2}_{i_1,i_2}$. Interprocessor *residual* transfers can be avoided altogether, basing all residuals on available solution values, even when they are not genuine.

A more radical approach, however, for reducing the amount of transferred data is described next.

6. Avoiding Finest Level Data Transfers

6.1. The τ correction and segmental refinements. We will now show that in the above FMG algorithms the *amount* of data transferred at each stage can drastically be reduced by omitting from the transfer any data related to the finest levels. This is based on a different view of the fine grid role in the FAS formulation.

Using (3.5), (3.8) and the definition of r^k, Eq. (3.4) can be rewritten in the form

$$L^{k-1}u^{k-1} = f^{k-1} + \tau_k^{k-1}, \tag{6.1}$$

where

$$\tau_k^{k-1} = L^{k-1}(\widehat{I}_k^{k-1}\widetilde{u}^k) - I_k^{k-1}L^k\widetilde{u}^k. \tag{6.2}$$

Without the τ_k^{k-1} term, Eq. (6.1) is the *original* coarse grid equation (3.7). Thus, τ_k^{k-1} expresses the correction introduced to the level-$(k-1)$ equation due to the "visit" to the finer level k.

Indeed this is a typical *defect correction* (cf., e.g., [13]): τ_k^{k-1} expresses the "defect" in L^{k-1}, i.e., the difference between it and the more accurate discretization L^k, that difference being measured on the current solution. Due to the defect correction, the solution of the level-$(k-1)$ equation (6.1) will have level-k accuracy. Indeed, at convergence $u^{k-1} = \widehat{I}_k^{k-1}u^k$, since $r^k = 0$.

Now observe that the "visit" to level k in the FMG_k algorithm includes only inter-grid transfers and level-k relaxation, all of which are purely local processes: they can be done on a *piece* of Ω^k at a time. Similarly, only a piece of Ω^{k-1} is needed at a time, since its role in correcting the level-$(k-2)$ equations is purely local. And so on.

Such observation lead to "*segmental refinement*" algorithms, proposed already in [2, Sec. 7.5], which use overlapping subdomains without any data transferred at the finest levels. These are actually domain decomposition algorithms, although their original motivation was to obtain a single-processing multigrid solver which uses storage area much smaller than the size of the finest grid, without using external storage.

Our general approach to multigrid domain decomposition will be to operate segmental-refinement-type procedures at several of the finest levels, with the procedures of Section 5 above operating at the coarser levels. For the purpose of clarity, however, we investigate first simpler algorithms that operate segmental refinement at *all* levels except for the coarsest level 1. On this coarsest level, which for these investigation purposes may itself be rather fine (cf. Sec. 5.2 above), the equations are always assumed to be solved *exactly*, by whatever method and processors.

Thus, in the algorithms studied in this section, there will be no interprocessor data transfer to and from levels $2, 3, \ldots, M$. The only communication between the P processors will be their interaction with a common coarsest level. The subgrid of Ω^k in which processor p operates is $\widehat{\Omega}_p^k = \Omega_p^{k,J(k)}$, defined as in Sec. 5.1, and the value of the current approximate solution which the p processor has at a gridpoint $i \in \widehat{\Omega}_p^k$ is denoted $\widetilde{u}_i^{k,p}$.

6.2. Two-level model case. To understand in detail the working of the algorithm, consider first the simpler two-level two-processor case. Taking the border line $\Omega_1 \cap \Omega_2$ to be the axis $\{x_1 = 0\}$, the gridlines of Ω_1^2 and Ω_2^2 will be those with $i_1 \leq 0$ and $i_1 \geq 0$, respectively. The gridlines of the extended subgrids $\widehat{\Omega}_1^2$ and $\widehat{\Omega}_2^2$ will be those with $i_1 \leq J$ and $i_1 \geq -J$, respectively. The FMG_2 algorithm for solving

$$L^2u^2 = f^2 \tag{6.3}$$

will consist in this case of the following 7 steps.

(A) Solve, by whatever method, the equation

$$L^1 u^1 = f^1 = I_2^1 f^2, \tag{6.4}$$

and set up the first approximation to the solution of (6.1) at a point $i \in \widehat{\Omega}_p^2$ to be $\tilde{u}_i^{2,p} = (\Pi_2^1 u^1)_i$. At this stage $\tilde{u}_i^{2,1} = \tilde{u}_i^{2,2}$ at all points $i = (i_1, i_2)$ of the overlap $\{-J \le i_1 \le J\}$. The solution interpolation Π_2^1 is bicubic.

(B) Perform ν relaxation sweeps, each processor p relaxing its own subgrid $\widehat{\Omega}_p^2$. At this stage the values of $\tilde{u}^{2,1}$ and $\tilde{u}^{2,2}$ in the overlap start to differ, because the "*interior boundary*" values $\tilde{u}_{J,i_2}^{2,1}$ and $\tilde{u}_{-J,i_2}^{2,2}$ cannot be relaxed, for lack of neighboring values within the same processor.

(C) Each processor p transfers to the coarse grid solution values and weighted-residual values needed to form both the modified injection

$$(\widehat{I}_2^1 \tilde{u}^2)_i = \begin{cases} \tilde{u}_{2i}^{2,1} & : i_1 < 0 \\ \tilde{u}_{2i}^{2,2} & : i_1 > 0 \\ \frac{1}{2}(\tilde{u}_{2i}^{2,1} + \tilde{u}_{2i}^{2,2}) & : i_1 = 0 \end{cases} \tag{6.5}$$

and the modified residual weighting $I_2^1 \tilde{r}^2$, defined by (3.3) with:

$$\tilde{r}_i^2 = \begin{cases} f_i^2 - L_2 u_i^{2,1} & : i_1 < 0 \\ f_i^2 - L^2 u_i^{2,2} & : i_1 > 0 \\ f_i^2 - \frac{1}{2}(L^2 u_i^{2,1} + L^2 u_i^{2,2}) & : i_1 = 0. \end{cases} \tag{6.6}$$

(D) On the coarse grid, calculate $\tilde{f}^1$ by (3.5), then solve the equation $L^1 \tilde{u}^1 = \tilde{f}^1$ *exactly*, by whatever method.

(E) Each processor p corrects its approximate solution $\tilde{u}^{2,p}$ by

$$\tilde{u}^{2,p} \leftarrow \tilde{u}^{2,p} + I_1^2(\tilde{u}^1 - \widehat{I}_2^1 \tilde{u}^{2,p}), \qquad (p = 1, 2). \tag{6.7}$$

Note that the injection $\widehat{I}_2^1 \tilde{u}^{2,p}$ used here and the modified injection $\widehat{I}_2^1 \tilde{u}^2$ defined by (6.6) are different in the "margins", where the marginal points for $p = 1$ are the gridpoints $i = (i_1, i_2)$ with $0 \le i_1 \le J$, and for $p = 2$ they are those with $-J \le i_1 \le 0$. Note further that after the correction (6.7), the solution values in the two processors agree at points corresponding to the coarse grid: $\tilde{u}_{2j}^{2,1} = \tilde{u}_{2j}^{2,2}$; but at other points they may well differ: $\tilde{u}_i^{2,1} \ne \tilde{u}_i^{2,2}$ for $i = (i_1, i_2)$ with either i_1 or i_2 or both being odd.

(F) Perform ν' additional relaxation sweeps.

(G) Repeat Steps (B) through (F) as many times as needed.

Usually, Step (G) could be omitted: just one cycle would be enough to make the algebraic error $\|\tilde{u}^2 - u^2\|$ substantially smaller than the discretization error $\|u^2 - I^2 u\|$, where $\tilde{u}^2$ are the *genuine* calculated values (see Sec. 5.1) and u^2 and u are the exact solution of (2.1) and (6.3), respectively. Here, however, we intend to investigate the questions: how small can $\|\tilde{u}^2 - u^2\|$ be made by additional cycles, and at what rate per cycle? How does this performance depend on the overlap

parameter J? Understanding these issues is important for the real *multi*level calculations, where much smaller discretization errors are to be reached.

6.3. Two-level mode analysis. For simplicity we assume J to be even. Note that the cycles (Step (G)) are stationary (introduce no change to $\tilde{u}^{2,1}$ and $\tilde{u}^{2,2}$) when the following two conditions are fulfilled: first, the difference equations are satisfied throughout the interior of each subgrid:

$$L^2\tilde{u}^{2,p}_{i_1,i_2} = f^2_{i_1,i_2}; \qquad \big((p=1,\ i_1 < J) \text{ and } (p=2,\ i_1 > -J)\big) \tag{6.8a}$$

and secondly, $\hat{I}^1_2 u^{2,1}$ and $\hat{I}^1_2 u^{2,2}$ coincide at the internal boundaries, i.e.,

$$\tilde{u}^{2,1}_{i_1,i_2} = \tilde{u}^{2,2}_{i_1,i_2} \qquad (i_1 = \pm J,\ i_2 \text{ is even}). \tag{6.8b}$$

These conditions leave unspecified the other values on the interior boundaries ($\tilde{u}^{2,1}_{J,i_2}$ and $\tilde{u}^{2,2}_{-J,i_2}$ for i_2 odd), so they can differ arbitrarily much from the corresponding genuine values. If these differences are large enough they can significantly infect the solution everywhere. Thus, there are arbitrarily bad solutions unaltered by the cycles.

Such bad solutions, however, will not be produced by the above FMG_2 algorithm. The values on the internal boundaries (e.g., $\tilde{u}^{2,1}_{J,i_2}$) differ from the corresponding genuine values ($\tilde{u}^{2,2}_{J,i_2}$) only because the latter are changed by relaxation, while the former are not. Moreover, any change in the genuine values ($\tilde{u}^{2,2}_{J,i_2}$) which is smooth (as a function of i_2) will be transmitted to the interior boundary ($\tilde{u}^{2,1}_{J,i_2}$) by the correction (6.4). Hence, the error left on the interior boundary will mainly be composed of high-frequency components of the form

$$\tilde{u}^{2,1}_{J,i_2} - \tilde{u}^{2,2}_{J,i_2} = A_\theta e^{i\theta i_2}, \qquad \left(\frac{\pi}{2} \le |\theta| \le \pi\right) \tag{6.9}$$

(the first i in the exponent denoting $\sqrt{-1}$), and their amplitude A_θ will be comparable to the size of the high frequency components in $u^2 - \Pi^2_1 u^1$. This size itself is usually (with a proper choice of Π^2_1) comparable to the discretization error $\|u^1 - I^1 u\|$, which roughly equals $2^s\|u^2 - I^2 u\|$, where s is either the smoothness order of u or the discretization order, whichever is smaller.

The error (6.9) is not itself an error in genuine values of $\tilde{u}^{2,1}$. But it induces an error $e^{2,1} = \tilde{u}^{2,1} - u^2$ throughout $\hat{\Omega}^2_1$, which vanishes on the real boundaries and satisfies

$$L^2 e^{2,1}_{i_1,i_2} = 0, \qquad (i_1 < J) \tag{6.10a}$$

and

$$e^{2,1}_{J,i_2} = A_\theta e^{i\theta i_2}. \tag{6.10b}$$

Since the effects of such high frequency boundary error is mainly local, we can approximate them by assuming (6.10) to hold in the semi-infinite domain

$\{i: -\infty < i_1 < J,\ -\infty < i_2 < \infty\}$, from which it follows, in case L^2 is the five-point Laplacian, that

$$e^{2,1}_{i_1,i_2} = \lambda(\theta)^{J-i_1} A_\theta e^{i\theta i_2}, \tag{6.11}$$

where $\lambda(\theta)$ satisfies $\lambda(\theta) + \lambda(\theta)^{-1} + 2\cos\theta - 4 = 0$ and $|\lambda(\theta)| < 1$. Hence, in the range (6.9) of high-frequencies,

$$|\lambda(\theta)| \le \lambda(\pi/2) = .268 \approx 2^{-2}.$$

The error for *genuine* values resulting from the decomposition are errors $e^{2,1}_{i_1,i_2}$ in the region $i_1 \le 0$. By (6.11) and the above estimates for $|A_\theta|$ and $|\lambda(\theta)|$, this decomposition error E_{dec}, in any norm, is roughly bounded by

$$E_{\text{dec}} \le 2^{s-2J} \|u^2 - I^2 u\|. \tag{6.12}$$

It follows that, for E_{dec} to be smaller than the discretized error $\|u^2 - I^2 u\|$, it is required that $J \ge s/2$. Thus, for a second order discretization, an overlap parameter $J = 2$ should be more than adequate.

6.4. Two-level numerical experiments precisely support this analysis. They were conducted with the above model problem in the 8×8 square, with the exact solution $u = \cos\big(A(x_1 - 4) + B(x_2 - 4)\big)$. The results for various values of A and B and various algorithm parameters (ν, ν', J) are shown in Table 2. The meshsize shown (h_2) is that of the fine grid: thus, for example, for $h_2 = .25$ the fine grid has 32×32 intervals, and for $h_2 = .0625$ it is 128×128. Using two different norms, the discrete L_2 and the L_∞ (maximum) norms, the table compares the algebraic error $\|\tilde{u}^2 - u^2\|$ with the discretization error $\|u^2 - I^2 u\|$. The algebraic errors were calculated at the end of cycles, i.e., following Step (F), except for those shown in the first column (marked 0), which are the algebraic errors after Step (A).

After enough cycles the algebraic errors tend to stationary values, shown in the last column of Table 2 (marked ∞). These errors are the decomposition errors E_{dec}. The calculated values clearly satisfy (6.12) (for $s = 2$) in all interesting cases. (A slight exception is seen for the highest frequency component $(A, B) = (1, 12)$, but it would disappear if the right-hand side were properly averaged: see the comment at the end of Sec. 5.4.)

Moreover, the table shows that in all cases the algebraic error after one cycle is already much smaller than the discretization error, and it is nearly independent of the overlap parameter J. In fact, for small J, the algebraic error after just one cycle is sometimes even smaller than that obtained after additional cycles: presumably it requires additional cycles for the interior-boundary error (6.9) to fully affect the *genuine* values (e.g., the values of $\tilde{u}^{2,1}_{i_1,i_2}$ at $i_1 \le 0$).

Notice that the *relative* decomposition error (relative to the discretization error) is larger in the L_∞ norm than in the L_2 norm. This is because, as predicted by (6.11), the error created by the decomposition is mostly concentrated at a restricted region — near the borderline — while the discretization error is not. Also conforming with (6.11) is the fact that the ratio between the L_2 and the L_∞ decomposition error norms is nearly independent of either h_2 or J.

6.5. Extension to more levels. There are a number of ways, currently under investigation, in which the above two-level ($M = 2$) algorithm can be extended to more levels ($M > 2$). The fundamental aspect in all of them is that the overlap parameter at level k, $J(k)$, should actually grow with $M-k$. Indeed, the accuracy one wants to reach is $O(\|u^M - I^M u\|$, while the error at the interior boundary of $\widehat{\Omega}_p^k$ is $O(\|u^{k-1} - I^{k-1}u\|) = 2^{s(M-k+1)}O(\|u^M - I^M u\|)$, hence, by the analysis of Sec. 6.3, it is necessary that

$$2J(k) \geq s(M-k+1). \tag{6.13}$$

Although the number of lines in the overlap implied by (6.13) increases with $M-k$, the number of *gridpoints* actually decreases, nearly geometrically. Hence, the total number of gridpoints required in the overlaps at all levels is still proportional to the number of interface gridpoints (gridpoints on the borderlines or border planes). Hence, the overlapping increases the amount of calculation only by a small fraction. We emphasize that the overlaps do not imply interprocessor data transfer: The algorithm avoids any such transfer at all levels except for the coarsest.

Preliminary experiments with $M \leq 4$ show that, with overlaps somewhat larger than (6.13), the algebraic errors indeed remain smaller than the discretization errors.

REFERENCES

[1] Alcouffe, R.E., A. Brandt, J.E. Dendy, Jr. and J.W. Painter, The multi-grid methods for the diffusion equation with strongly discontinuous coefficients, SIAM J. Sci. Stat. Comp. **2** (1981) 430–454.

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

[3] Brandt, A., Multi-grid solvers on parallel computers, in *Elliptic Problem Sovlers* (M. Schultz, ed.), Academic Press, New York, 1981, pp. 39–84.

[4] Brandt, A., *Multigrid Techniques: 1984 Guide, with Applications to Fluid Dynamics*, 191 pages, 1984, ISBN-3-88457-081-1. GMD Studien Nr. 85. Available from GMD-AIW, Postfach 1240, D-5205, St. Augustin 1, W. Germany, 1984.

[5] Brandt, A., Rigorous local mode analysis of multigrid, in *Preliminary Proc. 4th Copper Mountain Conf. on Multigrid Methods*, Copper Mountain, Colorado, April, 1989. An updated version can be obtained from the author.

[6] Brandt, A., Multilevel computations of integral transforms and particle interactions with oscillatory kernels, Comp. Phys. Comm. **65** (1991) 24–38.

[7] Brandt, A., Multigrid methods in lattice field computations, Nucl. Phys. B (Proc. Suppl.) **26** (1992) 137–180.

[8] Brandt, A. and A.A. Lubrecht, Multilevel matrix multiplication and fast solution of integral equations, J. Comp. Phys. **90** (1990) 348–370.

[9] Brandt, A. and V. Mikulinsky, Multigrid treatment of problems with highly oscillating boundary and boundary conditions, SIAM J. Sci. Stat. Comp., submitted.

[10] Brandt, A. and I. Yavneh, On multigrid solution of high-Reynolds incompressible entering flows, J. Comp. Phys. **101** (1992) 151–164.

[11] Brandt, A. and I. Yavneh, Accelerated multigrid convergence for high-Reynolds recirculating flows, J. Comp. Phys., in press.

[12] Mikulinsky, V., Multigrid treatment of boundary and free-boundary conditions, Ph.D. Thesis, Weizmann Institute of Science, 1992.

[13] Stetter, H.J., The defect correction principle and discretization methods, Num Math. **29** (1978), 425–443.

Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot 76100, Israel
E-mail address: mabrandt@weizmann.weizmann.ac.il

Table 1: Numerical results described in Sec. 5.4

								algebraic error					
								before cycle		cycle I		cycle II	
A	B	M	disc. err.	ν_0	ν_2	J	n	1	2	3	4	5	6
1	1	4	$6.72 \cdot 10^{-3}$	2	2	2	1	$3.03 \cdot 10^{-2}$	$2.95 \cdot 10^{-2}$	$4.02 \cdot 10^{-3}$	$3.78 \cdot 10^{-3}$	$5.63 \cdot 10^{-4}$	$5.26 \cdot 10^{-4}$
1	1	5	$1.68 \cdot 10^{-3}$	2	2	2	1	$8.77 \cdot 10^{-3}$	$8.71 \cdot 10^{-3}$	$1.21 \cdot 10^{-3}$	$1.19 \cdot 10^{-3}$	$1.80 \cdot 10^{-4}$	$1.76 \cdot 10^{-4}$
1	1	6	$4.20 \cdot 10^{-4}$	2	2	2	1	$2.42 \cdot 10^{-3}$	$2.41 \cdot 10^{-3}$	$3.41 \cdot 10^{-4}$	$3.40 \cdot 10^{-4}$	$5.19 \cdot 10^{-5}$	$5.16 \cdot 10^{-5}$
1	1	6	$4.20 \cdot 10^{-4}$	2	3	2	1	$2.06 \cdot 10^{-3}$	$2.05 \cdot 10^{-3}$	$2.20 \cdot 10^{-4}$	$2.19 \cdot 10^{-4}$	$2.46 \cdot 10^{-5}$	$2.44 \cdot 10^{-5}$
1	1	6	$4.20 \cdot 10^{-4}$	2	2	4	1	$2.42 \cdot 10^{-3}$	$2.41 \cdot 10^{-3}$	$3.40 \cdot 10^{-4}$	$3.38 \cdot 10^{-4}$	$5.13 \cdot 10^{-5}$	$5.11 \cdot 10^{-5}$
1	1	6	$4.20 \cdot 10^{-4}$	0	2	4	1	$2.64 \cdot 10^{-3}$	$2.64 \cdot 10^{-3}$	$3.77 \cdot 10^{-4}$	$3.76 \cdot 10^{-4}$	$5.75 \cdot 10^{-5}$	$5.72 \cdot 10^{-5}$
1	1	6	$4.20 \cdot 10^{-4}$	0	2	2	1	$2.65 \cdot 10^{-3}$	$2.65 \cdot 10^{-3}$	$3.80 \cdot 10^{-4}$	$3.79 \cdot 10^{-4}$	$5.83 \cdot 10^{-5}$	$5.80 \cdot 10^{-5}$
1	1	6	$4.20 \cdot 10^{-4}$	2	2	2	2	$1.35 \cdot 10^{-3}$	$1.35 \cdot 10^{-3}$	$1.81 \cdot 10^{-4}$	$1.81 \cdot 10^{-4}$	$2.66 \cdot 10^{-5}$	$2.65 \cdot 10^{-5}$
1	1	6	$4.20 \cdot 10^{-4}$	2	1	2	1	$3.25 \cdot 10^{-3}$	$3.24 \cdot 10^{-3}$	$6.76 \cdot 10^{-4}$	$6.74 \cdot 10^{-4}$	$1.58 \cdot 10^{-4}$	$1.57 \cdot 10^{-4}$
1	1	6	$4.20 \cdot 10^{-4}$	0	2	2	2	$1.36 \cdot 10^{-3}$	$1.36 \cdot 10^{-3}$	$1.83 \cdot 10^{-4}$	$1.83 \cdot 10^{-4}$	$2.70 \cdot 10^{-5}$	$2.69 \cdot 10^{-5}$
1	1	6	$4.20 \cdot 10^{-4}$	0	1	2	1	$3.82 \cdot 10^{-3}$	$3.82 \cdot 10^{-3}$	$8.12 \cdot 10^{-4}$	$8.11 \cdot 10^{-4}$	$1.94 \cdot 10^{-4}$	$1.93 \cdot 10^{-4}$
1	25	3	$3.17 \cdot 10^{3}$	2	2	2	1	$1.88 \cdot 10^{1}$	$1.71 \cdot 10^{1}$	1.881	1.425	$2.02 \cdot 10^{-1}$	$1.48 \cdot 10^{-1}$
1	25	4	8.140	2	2	2	1	$1.39 \cdot 10^{1}$	4.088	$8.92 \cdot 10^{-1}$	$7.22 \cdot 10^{-1}$	$1.40 \cdot 10^{-1}$	$1.21 \cdot 10^{-1}$
1	25	5	1.346	2	2	2	1	5.128	$6.84 \cdot 10^{-1}$	$5.06 \cdot 10^{-1}$	$8.29 \cdot 10^{-2}$	$3.42 \cdot 10^{-2}$	$1.21 \cdot 10^{-2}$
1	25	6	$3.10 \cdot 10^{-1}$	2	2	2	1	$5.02 \cdot 10^{-1}$	$2.16 \cdot 10^{-1}$	$7.15 \cdot 10^{-2}$	$4.93 \cdot 10^{-2}$	$9.78 \cdot 10^{-3}$	$8.88 \cdot 10^{-3}$
1	25	6	$3.10 \cdot 10^{-1}$	2	3	2	1	$4.90 \cdot 10^{-1}$	$2.01 \cdot 10^{-1}$	$6.50 \cdot 10^{-2}$	$4.52 \cdot 10^{-2}$	$6.45 \cdot 10^{-3}$	$6.16 \cdot 10^{-3}$
1	25	6	$3.10 \cdot 10^{-1}$	0	2	2	1	$8.12 \cdot 10^{-1}$	$8.12 \cdot 10^{-1}$	$5.34 \cdot 10^{-1}$	$2.25 \cdot 10^{-1}$	$7.48 \cdot 10^{-2}$	$5.01 \cdot 10^{-2}$
1	25	6	$3.10 \cdot 10^{-1}$	2	1	2	1	$5.76 \cdot 10^{-1}$	$3.16 \cdot 10^{-1}$	$1.39 \cdot 10^{-1}$	$1.07 \cdot 10^{-1}$	$3.98 \cdot 10^{-2}$	$3.21 \cdot 10^{-2}$
1	25	6	$3.10 \cdot 10^{-1}$	2	2	4	1	$5.01 \cdot 10^{-1}$	$2.16 \cdot 10^{-1}$	$7.13 \cdot 10^{-2}$	$4.92 \cdot 10^{-2}$	$9.66 \cdot 10^{-3}$	$8.78 \cdot 10^{-3}$
1	25	6	$3.10 \cdot 10^{-1}$	2	2	6	1	$5.01 \cdot 10^{-1}$	$2.16 \cdot 10^{-1}$	$7.13 \cdot 10^{-2}$	$4.92 \cdot 10^{-2}$	$9.66 \cdot 10^{-3}$	$8.77 \cdot 10^{-3}$
25	1	3	$3.17 \cdot 10^{3}$	2	2	2	1	$1.89 \cdot 10^{1}$	$1.72 \cdot 10^{1}$	1.882	1.427	$1.99 \cdot 10^{-1}$	$1.46 \cdot 10^{-1}$
25	1	4	8.140	2	2	2	1	$1.39 \cdot 10^{1}$	4.093	$8.82 \cdot 10^{-1}$	$7.04 \cdot 10^{-1}$	$1.37 \cdot 10^{-1}$	$1.18 \cdot 10^{-1}$

Table 1: (Cont.)

								algebraic error					
								before cycle		cycle I		cycle II	
A	B	m	disc. err.	ν_0	ν_2	J	n	1	2	3	4	5	6
25	1	5	1.346	2	2	2	1	5.117	$6.76 \cdot 10^{-1}$	$5.03 \cdot 10^{-1}$	$8.20 \cdot 10^{-2}$	$3.34 \cdot 10^{-2}$	$1.20 \cdot 10^{-2}$
25	1	6	$3.10 \cdot 10^{-1}$	2	2	2	1	$5.01 \cdot 10^{-1}$	$2.16 \cdot 10^{-1}$	$7.12 \cdot 10^{-2}$	$4.91 \cdot 10^{-2}$	$9.60 \cdot 10^{-3}$	$8.69 \cdot 10^{-3}$
25	1	6	$3.10 \cdot 10^{-1}$	2	3	2	1	$4.89 \cdot 10^{-1}$	$2.01 \cdot 10^{-1}$	$6.48 \cdot 10^{-2}$	$4.51 \cdot 10^{-2}$	$6.24 \cdot 10^{-3}$	$5.95 \cdot 10^{-3}$
25	1	6	$3.10 \cdot 10^{-1}$	0	2	2	1	$8.08 \cdot 10^{-1}$	$8.08 \cdot 10^{-1}$	$5.33 \cdot 10^{-1}$	$2.25 \cdot 10^{-1}$	$7.46 \cdot 10^{-2}$	$4.99 \cdot 10^{-2}$
25	1	6	$3.10 \cdot 10^{-1}$	2	1	2	1	$5.74 \cdot 10^{-1}$	$3.15 \cdot 10^{-1}$	$1.33 \cdot 10^{-1}$	$1.06 \cdot 10^{-1}$	$3.95 \cdot 10^{-2}$	$3.19 \cdot 10^{-2}$
25	1	6	$3.10 \cdot 10^{-1}$	2	2	4	1	$5.01 \cdot 10^{-1}$	$2.16 \cdot 10^{-1}$	$7.13 \cdot 10^{-2}$	$4.92 \cdot 10^{-2}$	$9.65 \cdot 10^{-3}$	$8.77 \cdot 10^{-3}$
25	1	6	$3.10 \cdot 10^{-1}$	2	2	6	1	$5.01 \cdot 10^{-1}$	$2.16 \cdot 10^{-1}$	$7.13 \cdot 10^{-2}$	$4.92 \cdot 10^{-2}$	$9.66 \cdot 10^{-3}$	$8.77 \cdot 10^{-3}$
25	25	6	$2.95 \cdot 10^{-1}$	2	2	2	1	1.469	$4.60 \cdot 10^{-1}$	$2.91 \cdot 10^{-1}$	$2.29 \cdot 10^{-1}$	$3.73 \cdot 10^{-2}$	$3.54 \cdot 10^{-2}$
25	1	6	$3.10 \cdot 10^{-1}$	2	2	2	2	$4.86 \cdot 10^{-1}$	$1.96 \cdot 10^{-1}$	$6.75 \cdot 10^{-2}$	$4.62 \cdot 10^{-2}$	$9.49 \cdot 10^{-3}$	$8.67 \cdot 10^{-3}$
1	100	6	8.49	2	2	2	1	$1.49 \cdot 10^{1}$	5.791	1.164	1.122	$2.36 \cdot 10^{-1}$	$2.31 \cdot 10^{-1}$
1	100	6	8.49	0	2	2	1	$1.49 \cdot 10^{1}$	$1.49 \cdot 10^{1}$	$1.48 \cdot 10^{1}$	5.686	1.092	1.052
100	1	6	8.49	2	2	2	1	$1.49 \cdot 10^{1}$	5.789	1.171	1.125	$2.35 \cdot 10^{-1}$	$2.30 \cdot 10^{-1}$
100	1	6	8.49	0	2	2	1	$1.49 \cdot 10^{1}$	$1.49 \cdot 10^{1}$	$1.48 \cdot 10^{1}$	5.685	1.100	1.055

Table 2: Numerical results described in Sec. 6.4

A	B	h_2	disc. err.	ν	ν'	J	algebraic error after cycles			
							0	1	2	∞
1	1	0.5	$1.09 \cdot 10^{-1}$	2	2	2	$4.07 \cdot 10^{-1}$	$1.61 \cdot 10^{-2}$	$8.51 \cdot 10^{-3}$	$8.39 \cdot 10^{-3}$
1	1	0.25	$2.70 \cdot 10^{-2}$	2	2	2	$8.72 \cdot 10^{-2}$	$2.19 \cdot 10^{-3}$	$1.49 \cdot 10^{-3}$	$1.49 \cdot 10^{-3}$
1	1	0.25	$2.70 \cdot 10^{-2}$	2	2	4	$8.72 \cdot 10^{-2}$	$1.96 \cdot 10^{-3}$	$1.41 \cdot 10^{-4}$	$1.52 \cdot 10^{-4}$
1	1	0.25	$2.70 \cdot 10^{-2}$	2	2	6	$8.72 \cdot 10^{-2}$	$1.96 \cdot 10^{-3}$	$6.16 \cdot 10^{-5}$	$1.26 \cdot 10^{-6}$
1	1	0.25	$2.70 \cdot 10^{-2}$	0	2	2	$8.72 \cdot 10^{-2}$	$6.22 \cdot 10^{-3}$	$1.05 \cdot 10^{-3}$	$1.49 \cdot 10^{-3}$
1	1	0.25	$2.70 \cdot 10^{-2}$	2	3	2	$8.72 \cdot 10^{-2}$	$2.08 \cdot 10^{-3}$	$1.55 \cdot 10^{-3}$	$1.50 \cdot 10^{-3}$
1	1	0.125	$6.72 \cdot 10^{-3}$	2	2	2	$2.06 \cdot 10^{-2}$	$1.73 \cdot 10^{-4}$	$2.05 \cdot 10^{-4}$	$2.13 \cdot 10^{-4}$
1	1	0.0625	$1.68 \cdot 10^{-3}$	2	2	2	$5.07 \cdot 10^{-3}$	$1.29 \cdot 10^{-5}$	$2.64 \cdot 10^{-5}$	$2.77 \cdot 10^{-5}$
1	1	0.0625	$1.68 \cdot 10^{-3}$	2	2	4	$5.07 \cdot 10^{-3}$	$9.67 \cdot 10^{-6}$	$3.42 \cdot 10^{-6}$	$4.66 \cdot 10^{-6}$
1	1	0.0625	$1.68 \cdot 10^{-3}$	2	2	8	$5.07 \cdot 10^{-3}$	$9.80 \cdot 10^{-6}$	$3.67 \cdot 10^{-8}$	$5.28 \cdot 10^{-9}$
1	1	0.0625	$1.68 \cdot 10^{-3}$	2	2	12	$5.07 \cdot 10^{-3}$	$9.80 \cdot 10^{-6}$	$3.63 \cdot 10^{-8}$	$3.87 \cdot 10^{-11}$
1	6	0.25	1.166	2	2	2	4.613	$8.00 \cdot 10^{-2}$	$9.96 \cdot 10^{-2}$	$9.99 \cdot 10^{-2}$
6	1	0.25	1.166	2	2	2	4.565	$2.68 \cdot 10^{-2}$	$1.33 \cdot 10^{-2}$	$3.23 \cdot 10^{-3}$
6	6	0.25	1.170	2	2	2	5.772	$6.01 \cdot 10^{-2}$	$7.77 \cdot 10^{-2}$	$7.76 \cdot 10^{-2}$
1	12	0.25	7.125	2	2	2	$1.34 \cdot 10^{1}$	$6.08 \cdot 10^{-1}$	9.659	9.717
1	12	0.25	7.125	2	2	4	$1.34 \cdot 10^{1}$	$1.59 \cdot 10^{-1}$	1.023	1.021
1	12	0.25	7.125	2	2	6	$1.34 \cdot 10^{1}$	$1.33 \cdot 10^{-1}$	$8.08 \cdot 10^{-3}$	$5.92 \cdot 10^{-3}$
12	1	0.25	7.125	2	2	2	$1.33 \cdot 10^{1}$	2.863	$7.38 \cdot 10^{-2}$	$5.89 \cdot 10^{-2}$
12	1	0.25	7.125	2	2	4	$1.33 \cdot 10^{1}$	$1.33 \cdot 10^{-1}$	$1.33 \cdot 10^{-2}$	$8.01 \cdot 10^{-3}$
12	1	0.25	7.125	2	2	6	$1.33 \cdot 10^{1}$	$1.33 \cdot 10^{-1}$	$5.96 \cdot 10^{-3}$	$2.67 \cdot 10^{-4}$

Table 2: (Cont.)

A	B	h_2	disc. err.	ν	ν'	J	algebraic error after cycles			
							0	1	2	∞
1	1	0.5	$2.57 \cdot 10^{-2}$	2	2	2	$1.17 \cdot 10^{-1}$	$5.39 \cdot 10^{-3}$	$4.62 \cdot 10^{-3}$	$4.40 \cdot 10^{-3}$
1	1	0.25	$6.40 \cdot 10^{-3}$	2	2	2	$2.19 \cdot 10^{-2}$	$7.64 \cdot 10^{-4}$	$9.24 \cdot 10^{-4}$	$9.28 \cdot 10^{-4}$
1	1	0.25	$6.40 \cdot 10^{-3}$	2	2	4	$2.19 \cdot 10^{-2}$	$4.69 \cdot 10^{-4}$	$7.84 \cdot 10^{-5}$	$8.67 \cdot 10^{-5}$
1	1	0.25	$6.40 \cdot 10^{-3}$	2	2	6	$2.19 \cdot 10^{-2}$	$4.69 \cdot 10^{-4}$	$1.79 \cdot 10^{-5}$	$5.61 \cdot 10^{-7}$
1	1	0.25	$6.40 \cdot 10^{-3}$	0	2	2	$2.19 \cdot 10^{-2}$	$1.51 \cdot 10^{-3}$	$4.76 \cdot 10^{-4}$	$9.28 \cdot 10^{-4}$
1	1	0.25	$6.40 \cdot 10^{-3}$	2	3	2	$2.19 \cdot 10^{-2}$	$7.90 \cdot 10^{-4}$	$9.54 \cdot 10^{-4}$	$9.28 \cdot 10^{-4}$
1	1	0.125	$1.60 \cdot 10^{-3}$	2	2	2	$4.96 \cdot 10^{-3}$	$6.60 \cdot 10^{-5}$	$1.24 \cdot 10^{-4}$	$1.28 \cdot 10^{-4}$
1	1	0.0625	$4.00 \cdot 10^{-4}$	2	2	2	$1.21 \cdot 10^{-3}$	$5.77 \cdot 10^{-6}$	$1.57 \cdot 10^{-5}$	$1.64 \cdot 10^{-5}$
1	1	0.0625	$4.00 \cdot 10^{-4}$	2	2	4	$1.21 \cdot 10^{-3}$	$2.17 \cdot 10^{-6}$	$2.02 \cdot 10^{-6}$	$2.68 \cdot 10^{-6}$
1	1	0.0625	$4.00 \cdot 10^{-4}$	2	2	8	$1.21 \cdot 10^{-3}$	$2.17 \cdot 10^{-6}$	$3.97 \cdot 10^{-8}$	$2.94 \cdot 10^{-9}$
1	1	0.0625	$4.00 \cdot 10^{-4}$	2	2	12	$1.21 \cdot 10^{-3}$	$2.17 \cdot 10^{-6}$	$3.97 \cdot 10^{-8}$	$2.23 \cdot 10^{-11}$
1	6	0.25	$3.29 \cdot 10^{-1}$	2	2	2	1.486	$1.12 \cdot 10^{-1}$	$1.19 \cdot 10^{-1}$	$1.19 \cdot 10^{-1}$
6	1	0.25	$3.29 \cdot 10^{-1}$	2	2	2	1.535	$1.25 \cdot 10^{-2}$	$5.76 \cdot 10^{-3}$	$3.14 \cdot 10^{-3}$
6	6	0.25	$2.34 \cdot 10^{-1}$	2	2	2	1.620	$8.33 \cdot 10^{-2}$	$8.73 \cdot 10^{-2}$	$8.74 \cdot 10^{-2}$
1	12	0.25	2.210	2	2	2	4.09	$4.00 \cdot 10^{-1}$	3.870	3.902
1	12	0.25	2.210	2	2	4	4.09	$6.08 \cdot 10^{-2}$	$3.84 \cdot 10^{-1}$	$3.85 \cdot 10^{-1}$
1	12	0.25	2.210	2	2	6	4.09	$5.71 \cdot 10^{-2}$	$3.10 \cdot 10^{-3}$	$2.05 \cdot 10^{-3}$
12	1	0.25	2.210	2	2	2	4.109	1.437	$4.89 \cdot 10^{-2}$	$4.52 \cdot 10^{-2}$
12	1	0.25	2.210	2	2	4	4.109	$5.82 \cdot 10^{-2}$	$6.66 \cdot 10^{-3}$	$4.78 \cdot 10^{-3}$
12	1	0.25	2.210	2	2	6	4.109	$5.71 \cdot 10^{-2}$	$3.00 \cdot 10^{-3}$	$1.52 \cdot 10^{-4}$