NASA Contractor Report 178319

ICASE REPORT NO. 87-37

ICASE

IMPLEMENTATION OF MULTIGRID METHODS FOR SOLVING NAVIER-STOKES EQUATIONS ON A MULTIPROCESSOR SYSTEM

Vijay K. Naik

Shlomo Ta'asan

Contract No. NAS1-18107
June 1987

INSTITUTE FOR COMPUTER APPLICATIONS IN SCIENCE AND ENGINEERING
NASA Langley Research Center, Hampton, Virginia 23665

Operated by the Universities Space Research Association

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

Implementation of Multigrid Methods for Solving Navier-Stokes Equations on a Multiprocessor System

Vijay K. Naik

Shlomo Ta'asan

Institute for Computer Applications in Science & Engineering

NASA, Langley Research Center, Hampton, Va 23665.

ABSTRACT

In this paper we present schemes for implementing multigrid algorithms on message based MIMD multiprocessor systems. To address the various issues involved, a nontrivial problem of solving the 2-D incompressible Navier-Stokes equations is considered as the model problem. Three different multigrid algorithms are considered. Results from implementing these algorithms on an Intel iPSC are presented.

Research supported by the National Aeronautics and Space Administration under NASA Contract No NAS1-18107 while the authors were in residence at ICASE, NASA Langley Research Center, Hampton, VA 23665.

1. Introduction

Multigrid algorithms are found to be optimal and efficient for solving a large class of problems involving partial differential equations on sequential machines [1]. Recently there has been increased interest in parallelizing these algorithms [Ref. 2-6, 13]. In most of the work that has been reported, simple problems such as the solution of Poisson equations are considered. Although these results are important for verifying the suitability of multigrid methods for parallel processing, they neither completely expose the data dependencies governing the communication costs nor do they bring out the difficulties involved in implementing real life problems - problems that will be solved on the future multiprocessor systems. The necessity for paying close attention to the data dependencies involved is further underlined by the fact that multigrid algorithms do not always perform optimally on the multiprocessor systems, even though the individual operations involved exhibit a high degree of parallelism [7]. To understand the limitations as well as the effectiveness of parallelizing the multigrid algorithms, we consider, in this paper, the problem of solving the the 2-D steady state incompressible Navier-Stokes equations. The solution of these equations represent some of the difficulties that are present in solving hard problems, but at the same time the equations are not too complex to experiment on the currently available multiprocessor systems. We also present methods for minimizing the communication costs on the architectures considered here.

In the following section we briefly describe the idea behind the multigrid methods and present three different algorithms based on this idea. In Section 3 the model problem is given. In Section 4 the implementation issues involved are discussed. Some performance results obtained by implementing the three multigrid algorithms on the Intel's Personal Supercomputer (iPSC) are described in Section 5. Conclusions are given in Section 6.

2. Multigrid Algorithms

Here we briefly describe the various multigrid techniques that are generally applied in solving partial differential equations. This discussion is by no means exhaustive, but is meant to illustrate the highlights of the multigrid methods. In Section 4 we describe methods of parallelizing these techniques.

Basic Idea

Consider a differential equation given by $\mathbf{L}U = F$ with boundary conditions $\mathbf{B}U = g$ defined on an n-dimensional domain in R^n. For simplicity of exposition let $\mathbf{L}$ be an elliptic operator. Let the difference scheme $\mathbf{L}^h U^h = F^h$ with boundary conditions $\mathbf{B}^h U^h = g^h$, approximate this differential equation. If this difference scheme is solved by relaxation, (Gauss-Seidel in lexicographic order, for instance), the error can be written as,

$$e_n^h = U^h - u_n^h \tag{1}$$

where, u_n^h is the current approximation after the n-th relaxation sweep. Now consider the ratio, $\mu_n = \|e_{n+1}^h\| / \|e_n^h\|$ where, $\|\cdot\|$ denotes the L_2-norm. From numerical experiments it is seen that the ratio increases with n from some value $\mu_0 < 1$ and approaches a number that may be very close to one. That is, convergence is fast in the first few steps and then slows down.

It is found that whenever the error e_n^h is not smooth μ_n is small, giving a good convergence rate. When e_n^h is smooth the resulting convergence rate becomes poor. That is, relaxation smoothes the error. The main idea of multigrid is this: if the error is smooth , approximate it by a coarse grid, say of mesh size $2h$.

Coarse Grid Correction

Let $\bar{u}^h$ be the approximation obtained on the fine grid after a few relaxation sweeps. The error $V^h = U^h - \bar{u}^h$ satisfies

$$\mathbf{L}^h V^h = r^h \tag{2}$$

where, $r^h = F^h - \mathbf{L}^h \bar{u}^h$. If V^h is a smooth function we can approximate it by a coarser grid function V^H that satisfies

$$\mathbf{L}^H V^H = r^H \tag{3}$$

where, $\mathbf{L}^H$ is a coarse grid approximation of $\mathbf{L}^h$ (e.g., a finite difference approximation on grid H to the same differential operator approximated by $\mathbf{L}^h$) and r^H is defined by,

$$r^H = \mathbf{I}_h^H r^h \tag{4}$$

where, $\mathbf{I}_h^H$ is a fine to coarse grid transfer operator (e.g., injection).

Note that the grid H has significantly fewer points than the grid h (one quarter of the points in two dimensions, if $H = 2h$) and hence, it is less expensive to solve Eqn. (3) than to solve Eqn. (2). Let $\bar{v}^H$ be the approximate solution obtained by solving Eqn. (3) by some method. We can use $\bar{v}^H$ to accelerate convergence of the fine grid by,

$$\bar{u}^h \leftarrow \bar{u}^h + \mathbf{I}_H^h V^H \tag{5}$$

where, $\mathbf{I}_H^h$ is an interpolation operator (e.g., linear interpolation). The process of calculating r^H, solving for V^H and interpolating the result to the fine grid is called the *coarse grid correction.*

The above described scheme, where equations corresponding to the error term V^h are solved on the coarse grid, is known as the *Correction Scheme* (CS). Instead of V^H, if we use $U^H = \bar{\mathbf{I}}_h^H \bar{u}^h + V^H$, where, $\bar{\mathbf{I}}_h^H$ is some fine to coarse grid transfer operator, as the unknown function on the coarse grid then the resulting scheme is known as the *Full Approximation Scheme* (FAS). The coarse grid function U^H approximates $\mathbf{I}_h^H U^h$, the *full* solution represented on the coarse grid. Such a scheme is useful when local refinement is needed or when nonlinear problems are to be solved. The FAS equations are

$$\mathbf{L}^H U^H = \bar{F}^H \tag{6}$$

where, $\bar{F}^H = \mathbf{L}^H \bar{\mathbf{I}}_h^H \bar{u}^h + \mathbf{I}_h^H r^h$. After solving the last equation approximately, we use the approximation $\bar{u}^H$ to correct the fine grid approximation as follows:

$$\bar{u}^h \leftarrow \bar{u}^h + I_H^h(\bar{u}^H - \bar{I}_h^H \bar{u}^h).$$

Multigrid Cycles

To solve Eqn. (3) or (6), if the underlying idea in the CS or the FAS is applied recursively, then one arrives at a general multigrid algorithm. These algorithms involve a cyclic order of computation and are referred to as the multigrid cycles. Different multigrid algorithms have been developed depending on the order and the frequency with which the grids are visited within a cycle. The most commonly used ones are the V and W cycles. In addition to these two types there is another type of multigrid cycle called the F cycle. For the sake of clarity, we will refer to the multigrid algorithms based on these three cycles as the V, W, and F algorithms, respectively. The recursive definitions of these algorithms are given in the Appendix. An important property of these multigrid algorithms is that, when the approximation scheme is appropriately chosen the rate of convergence of these methods can, in general, be made independent of the size of the problem. This is true for the F and W algorithms even if the

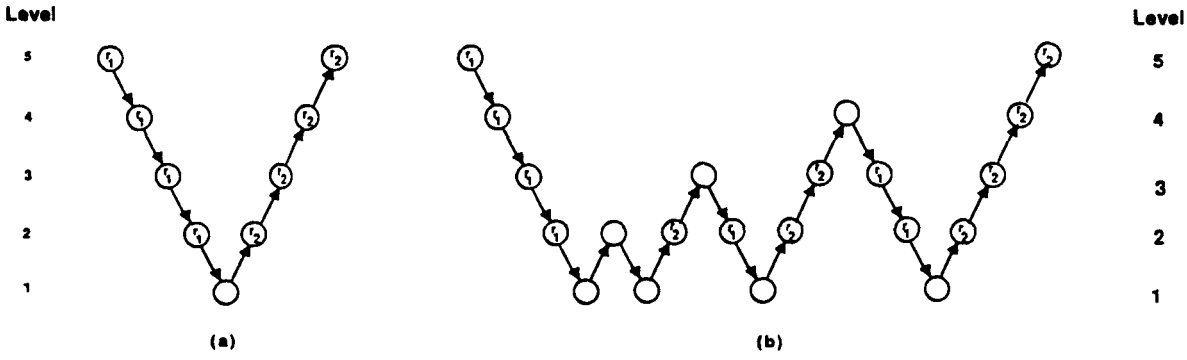


Figure 1 (a) V Cycle (b) F Cycle

problem is non-linear.

Figures 1 and 2 illustrate the order in which the V, F, and W algorithms visit different levels within a single cycle. Increasing level numbers correspond to finer mesh sizes. The letters r_1 and r_2 within the circles denote the number of relaxation sweeps on the corresponding levels. In all the three cases the values of r_1 and r_2 usually range from 0 to 2. On levels with empty circles $r_1 + r_2$ number of relaxations are performed. It can be easily verified that for the V algorithm each level is visited exactly once within a multigrid cycle. For the F algorithm, level i is visited $l - i + 1$ times where l is the highest level for that cycle. For the W algorithm, level i is visited 2^{l-i} times. Thus, the number of visits to the coarsest level grows exponentially for the W algorithm, whereas for the F algorithm it grows linearly.

Full MultiGrid (FMG) algorithms

To obtain a fast solution to the finest grid equations, a good initial approximation is needed on that level. A solution of the same problem on a coarser level can provide such an

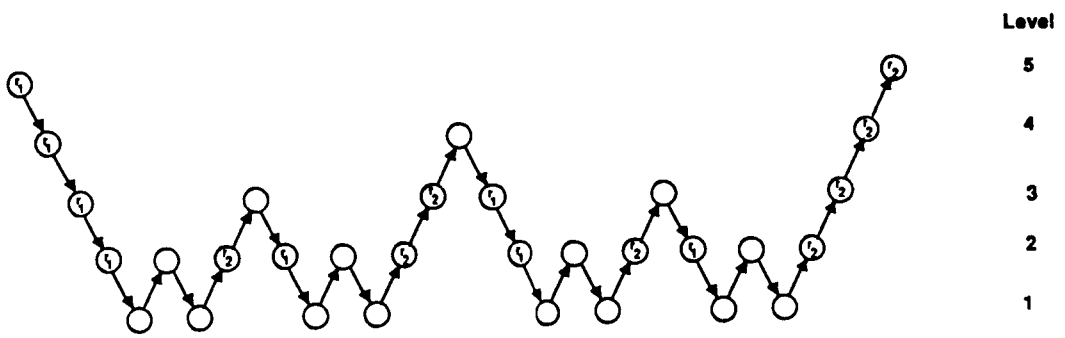


Figure 2 W Cycle

approximation. By applying the idea recursively one gets the *Full MultiGrid* (also referred to as *nested iteration*) algorithms for the three multigrid cycles described earlier. The FMG algorithm is defined in the Appendix.

In the following we first present the model problem and then describe its solution using the above described multigrid techniques.

3. Model Problem

The model problem considered here is that of solving the 2-D steady state incompressible Navier-Stokes equations. Such equations arise, for example, in studying the fully developed flow between two parallel plates where one plate may be moving with respect to the other plate. Their solutions present some of the difficulties involved in solving real life problems, but at the same time are simple enough for experimentation on currently available multiprocessor systems. Furthermore, all the three multigrid algorithms can be tested with these equations.

The equations in terms of vorticity ω and stream function ψ are:

$$\Delta\psi = \omega \tag{7a}$$

$$u\,\omega_x + v\,\omega_y = \frac{1}{\mathrm{Re}}\,\Delta\omega. \tag{7b}$$

Re is the Reynold's number of the fluid flow and u and v are the velocity components in the X and Y directions, respectively. The velocity components are given in terms of the stream function ψ by, $u = -\psi_y$, and $v = \psi_x$.

If the computational domain is $\Omega = \{(x, y) \mid 0 \le x \le 1,\ 0 \le y \le 1\}$ and if U_0 is the velocity of the moving plate, then the boundary conditions for such a flow are given by,

$u = v = 0$ at $y = 0$.
$u = U_0$, $v = 0$ at $y = 1$.
Periodicity is imposed in the X direction and
a constant pressure gradient is imposed on the flow.

In terms of ω and ψ the above boundary conditions are,

$$\psi = 0, \quad \frac{\partial \psi}{\partial y} = 0 \qquad \text{at} \quad y = 0 \tag{8a}$$

$$\psi = c, \quad \frac{\partial \psi}{\partial y} = -U_0 \qquad \text{at} \quad y = 1 \tag{8b}$$

where, c is a constant and ψ is periodic in the X direction.

Discretization

The Eqns. (7) are discretized as the following:

$$\frac{1}{h^2}\left[-4\psi_{i,j} + \psi_{i+1,j} + \psi_{i-1,j} + \psi_{i,j+1} + \psi_{i,j-1}\right] = \omega_{i,j} \tag{9a}$$

$$u_{i,j}\left[\frac{\omega_{i+1,j} - \omega_{i-1,j}}{2h}\right] + v_{i,j}\left[\frac{\omega_{i,j+1} - \omega_{i,j-1}}{2h}\right] = \frac{1}{h^2 \mathrm{Re}_h}\left[-4\omega_{i,j} + \omega_{i+1,j} + \omega_{i-1,j} + \omega_{i,j+1} + \omega_{i,j-1}\right] \tag{9b}$$

where,

$$u_{i,j} = \frac{1}{2h}\left[\psi_{i,j-1} - \psi_{i,j+1}\right],$$
$$v_{i,j} = \frac{1}{2h}\left[\psi_{i+1,j} - \psi_{i-1,j}\right],$$
$$\mathrm{Re}_h = \max\left\{\mathrm{Re},\ \beta \cdot h \max\left[|u_{i,j}|,\ |v_{i,j}|\right]\right\},$$
$0 \le i \le N$, $0 \le j \le N$, $(N-1)\cdot h = 1$, and β is a constant.

The discrete boundary conditions are as follows:

$$\psi_{0,j} = \psi_{N,j}, \quad \omega_{0,j} = \omega_{N,j} \qquad (j = 1, \cdots, N) \tag{10a}$$

$$\psi_{i,0} = 0, \quad \psi_{i,N} = c \tag{10b}$$

$$h\omega_{i,0} - \frac{2}{h}\cdot\left[\psi_{i,1} - \psi_{i,0}\right] = 0 \tag{10c}$$

$$h\omega_{i,N} - \frac{2}{h}\cdot\left[\psi_{i,N-1} - \psi_{i,N}\right] = 2U_0 \tag{10d}$$

For a derivation of Eqns. (10c) and (10d) see [10].

Multigrid Implementation

The above defined problem is solved using the FMG algorithms presented earlier. Here we describe some important aspects in applying the multigrid techniques.

Eqns. (9) are relaxed by Gauss-Siedel relaxation in a Red-Black ordering where at each color Eqn. 9b is relaxed first for $\omega_{i,j}$, followed by the relaxation of the ψ equation (Eqn. 9a). The relaxations of the boundary conditions (Eqns. 10c and 10d) are done in a different way. Note that a small change ($O(h^2)$) in ψ may result in a large change ($O(1)$) in ω. Hence a straightforward Gauss-Siedel relaxation of the boundary conditions (Eqns. 10c and 10d) is not appropriate; it may ruin the smoothness of ω obtained by the interior relaxation. The relaxation of the boundary conditions for ω should be such that it smoothes the residuals on the boundary and not make them zero as does the Gauss-Siedel relaxation. One way of achieving this is to change $\omega_{i,j}$ at the boundary such that the new residual at the point (i, j) is the average of the neighboring residuals, that is $r_{i,j}^{new} = \frac{1}{2}\left[r_{i+1,j}^{old} + r_{i-1,j}^{old}\right]$ [1]. On the coarsest grid, a straightforward Gauss-Siedel is applied for the boundary conditions as well.

To transfer only the smooth part of the residuals to the coarser level, in general a full-weighting operator is applied. Since the residuals after a red-black relaxation on the fine level are zero at half the points, we use a half-injection operator. A bi-linear interpolation operator

is applied during the interpolation within a given cycle and bi-cubic interpolation is used in obtaining a first approximation on any new level of the FMG algorithm. The necessity of employing the bi-cubic interpolation arises because of the following reasons. To solve the problem to the level of truncation errors in just a few cycles (that is, independent of the grid size) the initial residuals should behave like $O(h^2)$ so that in a few cycles we can reduce them to the level of truncation errors (which are also $O(h^2)$). The use of bi-linear interpolation will introduce high frequency errors of order h^2 on the new level that will have residuals of order one. Hence, if this interpolation is used, the number of multigrid cycles needed to solve the problem to the level of truncation errors ($O(h^2)$) will depend on h. Bi-cubic interpolation, on the other hand, introduces errors of order $O(h^4)$. Their residuals are of order $O(h^2)$ and therefore it is possible to get, in a few cycles, (independent of the mesh size) the level of truncation errors. For the definitions of full-weighting, injection, half-injection, bi-linear, and bi-cubic operators see [12].

4. Implementation on a Message Passing System

In this section we discuss some of the implementation aspects of the above described multigrid algorithms on message based MIMD multiprocessor systems. We consider the multiprocessor systems based on hypercube interconnection topology; but the principles developed here are applicable to other message passing systems as well. Experimental results from a specific message passing system are presented in the next section. A detailed discussion of the performance issues involved is given in [7]. Descriptions on the hypercube topology and some relevant properties can be found in [8], [11].

An efficient implementation of the multigrid algorithms on a local memory system involves partitioning the domain such that the computational load is distributed as evenly as

possible at all the times and the communication cost is kept minimum. It may not always be possible to achieve both at the same time and hence we try to reach a balance between the two. The partitioning scheme must take into account both the interior and the boundary points of the domain so that the load is evenly distributed, whereas the communication costs must take into account the communication parameters of the underlying architecture as well as the data dependencies involved in various operations. In general the interior points have a higher computational work associated with them whereas the boundary points have higher communication demands. Furthermore, for the type of equations we are solving, the boundary conditions drive the problem and so the order in which the interior and the boundary points are treated is extremely important. All these constraints must be taken into account by any partitioning scheme adopted for achieving good performance. In the following, we first describe the main implementation issues that include schemes for partitioning the domain on various levels and mapping these partitions on the individual processors of the hypercube multiprocessor system so that the total cost of a general multigrid algorithm is minimized. After that we describe methods that further reduce the communication costs for the problem being solved here.

Partitioning Schemes

For the sake of simplicity consider a 2-D square domain with $2^L \times 2^L$ mesh units on the highest level L of the multigrid algorithm being implemented. When boundary conditions are periodic, this implies that the number of interior points along that direction is also a power of two. Along the direction in which the boundary conditions are non-periodic the number of interior points is one less than a power of two. To describe the partitioning schemes we assume that the number of points is a power of two in both directions. We divide the domain on the finest level into 2^x partitions along the X direction and 2^y partitions along the Y direction. Thus we get 2^{x+y} partitions with each partition having 2^{L-x} points along the X direction

and 2^{L-y} points along the Y direction. Depending on the distribution of the work on various levels among the partitions one gets different partitioning schemes.

A *fixed region* partitioning strategy is the one where on the successive coarser levels each partition contains the same region of the domain, i.e., the region formed by the points that are the coarse level counterparts of points on the partition on the finest level. Under this partitioning scheme, with each coarsening the number of points associated with each partition decreases by a factor of four until the level $\max(x, y)$ is reached. Below level $\max(x, y)$ each partition has at most one line of points. With further coarsening the number of points per partition is halved until level $\min(x, y)$ is reached. On that level each partition has at most one point of the domain. Furthermore, in moving from level l to level $l-1$, where $\max(x, y) \geq l > \min(x, y)$, the number of partitions having any points and hence any computational work reduces by a factor of two. When l is less than or equal to $\min(x, y)$ this number reduces by a factor of four.

The parameters x and y in the above described scheme determine the *size* and the *shape* of the partitions of the domain on any given level. In general, the partition size determines the total computational work associated with a partition, whereas the partition shape affects the communication costs, since the values on the boundary of the adjacent partitions must be exchanged. For the fixed region partitioning scheme the shapes affect the overall distribution of work as well. Note that the shape of a partition is meaningful only on the levels where the number of points per partition is greater than one. A detailed discussion on the combined effect of the iteration stencil, the partition shape, and the communication parameters of the underlying architecture on the total communication costs for single grid algorithms is given by Reed et al. in [9]. Their discussion concentrates on minimizing the communication cost when the computational work is evenly distributed and it remains the same throughout the computation. For multigrid algorithms, the fact that the computational work decreases on the coarser

levels must also be taken into account. We explain this point with the help of an example. Consider a domain with 64 by 64 points on the fine level. Assume that 64 partitions are to be made on the fine level. In the fixed region partitioning scheme, if square partitions are used, then each partition has at least one point on levels 3, 4, 5, and 6. (Level 1 is the coarsest level.) On the other hand if one were to partition the domain in strips (one column of 64 points in each partition, for example), then only on level 6 would all partitions have some points assigned to them. Note that in both cases each partition has the same computational work on the finest level. Thus among squares, rectangles, and strips, squares balance the computational load best. In our implementations we assume that the partitions are square in shape on all grids.

In the fixed region partitioning scheme considered above, the regions of the domain are permanently assigned to the processors on all the levels even when the associated computational work is small. Sometimes it is advantageous to resort to a *shifting region* partitioning scheme. In this scheme below a certain level l^* the work on the entire domain is shifted to one node so that on all the successive coarser levels there is no communication cost. On levels l^* and above the computational work is uniformly distributed among all the partitions, but below level l^* the computation is serialized. Thus, every time there is transition between levels l^* and $l^* - 1$ either the data has to be gathered to one partition or scattered to all partitions from one partition. This scheme performs well if l^* is such that

$$\sum_{l=1}^{l^*-1} \left[C_{part_l} + C'_{part_l} \right] > \sum_{l=1}^{l^*-1} C_{Dom_l} + G_{l^*} + S_{l^*}$$

where, C_{part_l} and C'_{part_l} denote the computation and communication costs, respectively, associated with a partition on level l. C_{Dom_l} is the computation cost associated with the entire domain on level l. G_{l^*} and S_{l^*} are the costs of gathering and scattering the domain on level l^*, respectively.

In both the schemes described above, below a certain level some of the partitions do not contribute to any useful work, but for the fixed region scheme this occurs at a level below which there is not enough work to distribute among all the partitions, whereas for the shifting region scheme this may occur even when some work can be distributed but is not, in order to reduce the communication costs. The computational load is better balanced in the former case, but the computation and the communication load together may be balanced better for the later scheme. Although the later scheme promises better performance, generally accurate prediction of l^* is difficult. Furthermore, this scheme has a higher programming cost that must also be taken into account.

It is possible to further reduce the communication costs in the shifting region partitioning scheme by replicating the work for the entire domain below level l^* at each node. Here the cost of scattering the domain on level l^* is avoided. The gather operation can be performed in a tree fashion where each node acts as the root of a binary tree. Thus, the cost of gathering remains the same. So do the total computational costs. We refer to this scheme as the *modified shifting region* scheme.

Mapping

Depending on the interconnection network incorporated in the underlying architecture and depending on the parameters that determine the communication costs, partitions are mapped onto the processors so that the communication costs are minimized while the computational load is uniformly distributed. To illustrate the appropriate mapping techniques for hypercubes, we consider the fixed region partitioning scheme. The other two partitioning schemes have similar communication properties on levels l^* and above, and so the same mapping schemes can be applied. In the following discussion we assume that each node is assigned a single partition on each level.

On the fine level each partition has one or more points in each direction. Furthermore, the neighboring points of the domain reside on the same partition or on neighboring partitions. Thus, by mapping the partitions onto neighboring nodes one can minimize the communication costs on the fine level since most of the data dependencies are of the nearest neighbor type. As mentioned earlier with each successive coarsening the number of points associated with each partition decreases by a factor of four and on some level l each partition has only one point of the domain. Below this level with each coarsening not only the number of partitions having any points of the domain decreases by a factor of four, but also the distance between the partitions having the neighboring points keeps doubling. Thus, to avoid the increase in the communication costs on the lower levels the mapping scheme employed should be such that on any level the neighboring points are always on the neighboring partitions. Since in the hypercube topology each node has $\log N$ neighbors, where N is the total number of nodes, each partition has $\log N$ neighboring partitions and so one might expect to find such a mapping strategy. Unfortunately, it is not possible to map the partitions on the fine level so that on all the lower levels the neighboring partitions are found on the neighboring nodes without any remapping. For such mappings to be possible, the network should contain odd length cycles†, which the hypercube network does not support. But it is possible to map the partitions on the fine level so that up to and including the level l all neighboring partitions are on neighboring nodes and below that level the neighboring partitions are at most two hops away. This is achieved by making use of the binary reflected gray code scheme [3]. Note that in this scheme once the partitions are mapped onto the nodes, they remain stationary on the same node on all levels.

One way to avoid having to communicate to nodes that are two hops away below some level l is by remapping the partitions on levels below level l so that the distance between

† A cycle is a path formed by the edges connecting adjacent nodes such that path begins and ends at the same node. The length of the cycle is the number of edges forming the cycle.

neighboring partitions is again one hop. The exchange algorithm given in [3] performs such a remapping. With the exchange algorithm normal communication costs are reduced because the neighboring partitions are always one hop away, but on each level it has the extra overhead of the exchange operation. The complexity of the code is also higher. The overall gains are higher if a large number of relaxations are performed on each level, otherwise, it may not be worth the additional programming complexity. Furthermore, if the overhead of message initialization is significantly higher than the actual transmission cost then the difference in sending a message to a node one hop away or two hops away may not be high. For the MIMD architecture we are considering here, each processor is associated with substantial local memory and so several points of the finest level are assigned to each processor. In such cases the level l_m is small compared to L, the finest level. The message initialization costs are also higher than the actual transmission costs. So we do not consider the exchange algorithm.

Additional Communication Cost Considerations

The partitioning and the mapping strategies discussed so far take into account only the general structure of the multigrid algorithms. For extracting the best possible performance it is necessary to take a close look at the data dependencies of the individual multigrid operations such as relaxation, injection, and interpolation. Such an inspection allows one to have a better handle on the scheduling as well as on the frequency of the messages being transmitted. In addition to the data dependencies involved in the algorithm one must take into account the architecture dependent parameters such as the message initialization cost, message size, and per unit transmission cost. In the architecture that we are considering the message initialization cost is high as compared to the actual per byte transmission cost. The packet sizes are also large (1024 bytes). So the emphasis would be on reducing, whenever possible, the number of times the messages are being initialized by looking ahead and sending data that is needed in the

future. In the following, we describe how these considerations can be incorporated in the implementation schemes. The case of at most one point per partition is considered since it represents the worst case. When there are more points per partition, the communication requirements are less stringent. Note that some of the techniques may not be advantageous if the message initialization costs are low or when the message sizes are small.

As discussed earlier the relaxation process uses a five point stencil with red-black ordering. Here first the interior black and then the interior red points are treated. For the points along the boundary at $y = 0$ and $y = 1$ equations are solved only for ω and the amount of computation associated with a boundary point is smaller than that at an interior point. But to achieve the desired convergence rate the boundary points must be relaxed after relaxing the interior points during each iteration. Thus, a straightforward implementation may add two extra phases of communication during each iteration. Since the computational work is small, but the communication cost is high, the boundary points tend to become a bottleneck. This can be avoided if we couple the boundary points on a given grid with the nearest interior point. By coupling we mean the following. The boundary points alone, on any level do not form a partition. The partition that has the interior points belonging to the first and/or last row of the domain on a given grid contains the adjacent boundary points. If the partition to which the boundary points are assigned does not have any interior points on the next coarser level then the relevant boundary points are moved to the next interior partition on the coarser level. Because of the small amount of computation associated with the relaxation of the boundary points, the load imbalance is quite small. On the other hand the savings in communication cost are significant. In this scheme we relax the interior black points first and send the values of the black points and of the domain boundary red points (values of which are from the previous iteration) on the partition boundary to appropriate processors containing the neighboring partitions. After this the red interior points and then the domain boundary black points (if the

partition contains the domain boundary) are relaxed. Now both the values of the interior red points and the domain boundary black points are sent to the neighboring partitions. On receiving the values from the neighboring partitions the domain boundary red points (if any belong to the partition) are relaxed. This completes one iteration. Thus during each iteration messages are exchanged exactly twice everywhere, at the same time the desired order of relaxation for the interior and the boundary points is maintained.

The restriction operation may require one phase of communication among the appropriate partitions to completely define the residuals and to have all the necessary data for relaxing the black points on the coarse level. But this communication phase can be avoided by sending additional information at the end of relaxing red points just before coarsening. Here in addition to sending the new values of the red points on the boundary, we also send the values of the adjacent black points from the interior of the partition (or from the opposite partition if there is only one point per partition along that direction) to the appropriate neighboring partitions. Furthermore, this arrangement allows one to start the relaxation on the coarse level without any additional communication.

For analyzing the communication requirements of the interpolation process there are two points that should be noted. For simplicity we consider linear interpolation. First only the red points on the fine level need to be interpolated since immediately after the interpolation black points are relaxed. Secondly, the interpolated values of the red points are needed only at the black points during the relaxation after the interpolation. Thus, when each partition has at most one point (on coarse as well as fine level), the fine level red points can be interpolated at the partitions having black points and the relaxation on the fine level can begin without having to communicate to the red points. Some amount of information has to be transmitted to the partitions containing black points that were not present on the coarse level (only half of all the

fine level black points are present on the coarse level). It can be shown that during the relaxations after interpolation if one were to perform communication first and then computation, the communication phase in the interpolation process can be avoided even if there is at most one point per partition. This is true only for linear interpolation and simpler boundary conditions. For higher order interpolation or if the boundary conditions are complex as in our model problem, and if we couple the boundary points with the interior points, then at least one phase of communication is required during the interpolation process.

Thus, we have shown that by suitablely rearranging and combining the messages, usually the communication phases can be restricted to those during the relaxation process. In the next section we show some results obtained by applying these strategies in solving the model problem using multigrid algorithms on an Intel iPSC.

5. Results

The three multigrid algorithms discussed in Section 2 were implemented for solving the model problem on an Intel iPSC. The details of this architecture can be found in [8]. The experimental results presented here were obtained with the Release 3.0 iPSC operating system. Here we present the results of implementing both the fixed region and the shifting region partitioning schemes. Additional performance results can be found in [7].

The effect of the number of processors available to solve the problem to the level of discretization error on the execution time is shown in Figure 3. The figure also compares the parallelizability of the three algorithms studied. Clearly, the V algorithm takes the highest advantage of the added processors in reducing the total execution time, whereas the W algorithm gains the least. On the other hand, on a single processor the V algorithm performs poorly as compared to the other two algorithms. The three algorithms have different numerical

properties and hence different convergence rates. For the problem we are considering here, both the F and W algorithms need about three FMG cycles to solve a 128 x 128 problem to the level of discretization error, whereas the V algorithm takes about seven cycles to obtain the same accuracy. Over the range of the cube sizes and the problem sizes considered the F algorithm performs the best.

The effect of the problem size on the system efficiency is shown in Figure 4 for the F algorithm. The other two algorithms show similar trends. The efficiency of the system is computed using the time taken by the F algorithm to solve the problem on a single node. From Figure 4 it is clear that increasing problem size for a given number of nodes results in improvements in efficiency. This is expected because by increasing the problem size while keeping the number of nodes the same, the partition size assigned to each node increases resulting in a larger computation cost per node without increasing the communication cost.

In Figure 3 the three algorithms were compared using the absolute execution times. To compare the three algorithms qualitatively it is necessary to look at the relative efficiencies of the system for the three algorithms. As stated earlier, the algorithms have different numerical properties as well as different communication requirements. Hence, we compute the efficiencies using the best sequential timing (given by the F algorithm for our problem). We refer to such an efficiency as the normalized efficiency. In Figure 5, the normalized efficiencies of the three algorithms are compared as the dimension of the hypercube is varied. In all the cases the problem is solved to the same level of numerical accuracy. It can be seen that when the partition sizes are large or when the hypercube size is small, both F and W algorithms perform better than the V algorithm in spite of the adverse communication costs. For the problem and machine sizes considered here, the FMG F algorithm has the best overall performance. When partition sizes are small the V algorithm may perform better even though its

convergence properties are inferior.

The effect of employing the shifting region partitioning scheme on the performance is shown in Figure 6. Recall that in this scheme the work is serialized below some level l^* by moving all the regions of the domain to a single node. The percent increase in efficiency by serializing the work below levels 2 through 5 on a 16 node hypercube are shown for two problems having 64x64 and 128x128 points on the fine level. In this figure $l^* = 1$ corresponds to the fixed region partitioning scheme, i.e., no moving of regions takes place. It can be seen that the performance peeks out at a particular value of l^*. The savings in the communication costs achieved by serializing the work above this level is offset by the increase in the computation cost. Note that when the problem size is small or when the size of the partitions assigned to each processor is small, the gains are higher. Here each partition has a smaller piece of work on the highest level and so the communication costs are more dominant. By serializing the computation below a certain level, the percentage reduction in the total cost is higher than that in the larger size problems. In Figure 7 we show the effect of the above described partitioning scheme when the computing power is increased by adding more processors. Note that for the larger size hypercube, the cost of scattering and gathering the data is also higher. But now the computational work associated with each partition has decreased and so the communication costs form a higher proportion of the total cost.

6. Conclusions

Schemes for efficient implementation of three multigrid algorithms for solving 2-D incompressible Navier-Stokes equations on a message passing system are presented. It is shown that, the communication costs can be reduced even in such moderately difficult problems by simultaneously taking into account the data dependencies of the various operations as well as the communication parameters of the underlying architecture. Performance of the three

algorithms are compared by implementing the schemes on an Intel iPSC. It is found that the FMG algorithm based on the F cycle performs the best over the range of the problem and machine sizes considered.

ACKNOWLEDGEMENTS

We would like to thank the Mathematics and Statistics Research Section at Oak Ridge National Laboratory for granting us the use of their iPSC. We are particularly indebted to Tom Dunigan for his help during the course of the experimental work. We are grateful to Bob Voigt and Merrell Patrick for their continued support and encouragement.

REFERENCES

[1] A. Brandt, *Multigrid Techniques: 1984 Guide*, GMD Studien 85, Gesellschaft fur Mathematik und Datenverarbeitung, St. Augustin, 1984.

[2] B. Briggs, L. Hart, S. McCormick, and D. Quinlan, *Multigrid methods on a hypercube*, Proc. Third Copper Mountain Conference on Multigrid Methods, Copper Mountain, Colorado, 1987.

[3] T. F. Chan and Y. Saad, *Multigrid algorithms on the hypercube multiprocessor*, IEEE Trans. Comput., vol. 35, pp. 969-977, 1986.

[4] T. F. Chan and R. S. Tuminaro, *Design and implementation of parallel multigrid algorithms*, Proc. Third Copper Mountain Conference on Multigrid Methods, Copper Mountain, Colorado, 1987.

[5] G. M. Johnson and J. M. Swisshelm, *Multigrid for parallel-processing supercomputers*, Proc. Third Copper Mountain Conference on Multigrid Methods, Copper Mountain, Colorado, 1987.

[6] H. Mierendorff, *Parallelization of multigrid methods with local refinements for a class of non-shared memory systems*, Proc. Third Copper Mountain Conference on Multigrid Methods, Copper Mountain, Colorado, 1987.

[7] V. K. Naik and S. Ta'asan, *Performance studies of the multigrid algorithms implemented on hypercube multiprocessor systems*, Proc. Second Conference on Hypercube Multiprocessors, Knoxville, Tennessee, 1986.

[8] J. Rattner, *Concurrent processing: a new direction in scientific computing*, AFIPS Conference Proceedings, National Computer Conference, vol. 54, pp. 157-166, 1985.

[9] D. A. Reed, L. M. Adams, M. L. Patrick, *Stencils and problem partitionings: Their influence on the performance of multiple processor systems*, IEEE Trans. Comput., 1987, to appear.

[10] P. J. Roache, *Computational Fluid Dynamics*, Hermosa Publishers, 1972.

[11] Y. Saad and M. H. Schultz, *Topological properties of hypercubes*, Research Report, Yale University, YALEU/DCS/RR-389, 1985.

[12] K. Stuben and U. Trottenberg, *Multigrid methods: fundamental algorithms, model problem analysis and applications*, in MULTIGRID METHODS, W. Hachbusch and U. Trottenberg (eds.), lecture notes in mathematics, vol. 960, pp. 1-176, Springer-Verlag, 1982.

[13] C. Thole, *The SUPRENUM approach: MIMD architecture for multigrid algorithms*, Proc. Third Copper Mountain Conference on Multigrid Methods, Copper Mountain, Colorado, 1987.

Appendix

Consider a sequence of discretizations with mesh sizes $h_1 > h_2 > \cdots > h_M$, where, $h_k = 2 \cdot h_{k+1}$. Let the h_k-grid equations be,

$$\mathbf{L}^k U^k = F^k \tag{A.1}$$

where, $\mathbf{L}^k$ approximates $\mathbf{L}^{k+1}$ ($k < M$) (they all approximate some differential operator).

V and W Algorithms

Given an approximate solution $\bar{u}^l$, to Eqn. (A.1) on grid l, the V or W algorithms improve $\bar{u}^l$ by,

$$\bar{u}^l \leftarrow \text{MGVW}(\nu, l, \bar{u}^l, F^l).$$

The recursive definition of function MGVW() is given below. For $\nu = 1$ we get the V algorithm and $\nu = 2$ we get the W algorithm. Level 1 denotes the coarsest level and k is any level less than or equal to l.

```
MGVW( ν, k, ū^k, F^k)
  begin
    perform r_1 relaxation sweeps on grid k
      and store the new values in ū^k;

    if k > 1
      begin
        perform coarsening:
          begin
            ū^{k-1} ← Ī_k^{k-1} ū^k;
            F^{k-1} ← I_k^{k-1} [F^k − L^k ū^k] + L^{k-1} ū^{k-1};
          end coarsening
        for i = 1 until ν do
          ū^{k-1} ← MGVW( ν, k−1, ū^{k-1}, F^{k-1});

        perform correction interpolation:
          ū^k ← ū^k + I_{k-1}^k [ū^{k-1} − Ī_k^{k-1} ū^k];
      end

    perform r_2 relaxation sweeps on grid k
      and store the new values in ū^k;

    return ū^k;
  end MGVW.
```

F Algorithm

Given an approximate solution $\bar{u}^l$ to Eqn. (A.1) on grid l, the F algorithm improves $\bar{u}^l$ by,

$$\bar{u}^l \leftarrow \text{MGVF}(\nu, l, \bar{u}^l, F^l).$$

The function MGVF() is defined recursively below. As before level 1 denotes the coarsest level and k is any level less than or equal to l. When $\nu = 2$ we get the F algorithm and when $\nu = 1$ we get the V algorithm.

```
MGVF( ν, k, ū^k, F^k)
    begin
        perform r_1 relaxation sweeps on grid k
            and store the new values in ū^k;

        if k > 1
            begin
                perform coarsening:
                    begin
                        ū^{k-1} ← Ī_k^{k-1} ū^k;
                        F^{k-1} ← I_k^{k-1} [F^k − L^k ū^k] + L^{k-1} ū^{k-1};
                    end coarsening
                for i = 1 until ν−1 do
                    ū^{k-1} ← MGVF( ν, k−1, ū^{k-1}, F^{k-1});

                perform a V cycle:
                    ū^{k-1} ← MGVW( 1, k−1, ū^{k-1}, F^{k-1});

                perform correction interpolation:
                    ū^k ← ū^k + I_{k-1}^k [ū^{k-1} − Ī_k^{k-1} ū^k];
            end

        perform r_2 relaxation sweeps on grid k
            and store the new values in ū^k;

        return ū^k;
    end MGVF.
```

FMG Algorithm

If $\bar{u}^1$ is an approximate solution to Eqn. (A.1) on the coarsest grid, then a fast solution on level l is obtained by,

$$\bar{u}^l \leftarrow \text{FMG}(\, n,\, l,\, \bar{u}^1,\, F^1).$$

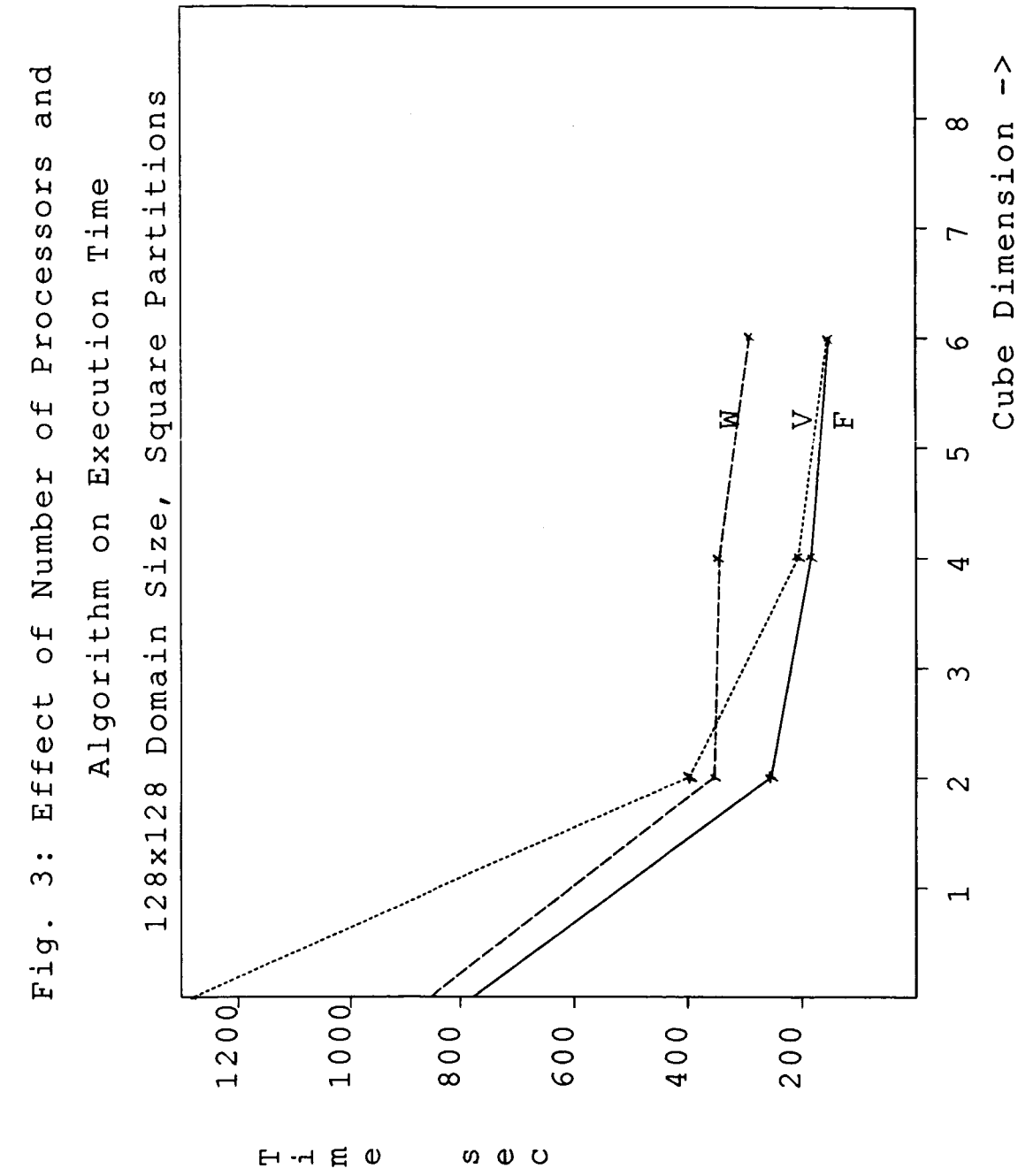
The definition of the FMG() algorithm is given below. In that definition MG() either MGVW() or MGVF() defined above. Depending on the appropriate selections of MG() and the value of the parameter ν, either V, F, or the W cycles are performed. Parameter n denotes the number of times a cycle is to be repeated at each level.

```
FMG( n,l,ū¹,F¹)
    begin
        solve for ū¹ by direct method
        or by several relaxation sweeps on grid 1;

        for k = 1 until l do
            begin
                perform interpolation:
                    ū^k ← Π^k_{k-1} ū^{k-1};

                for i = 1 until n do
                    ū^k ← MG( ν, k,ū^k,F^k);
            end
        return ū^l
    end FMG
```

Fig. 3: Effect of Number of Processors and Algorithm on Execution Time



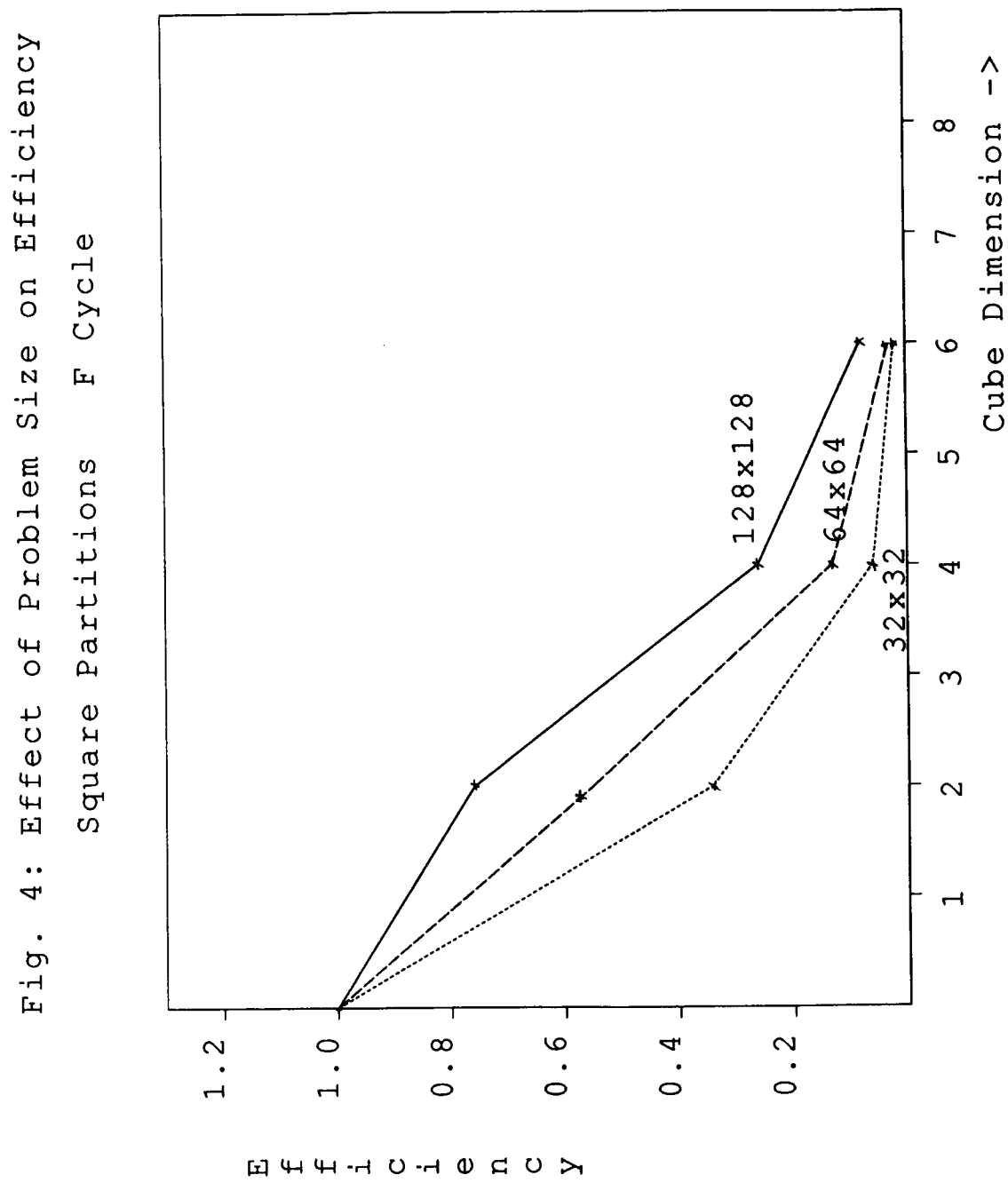
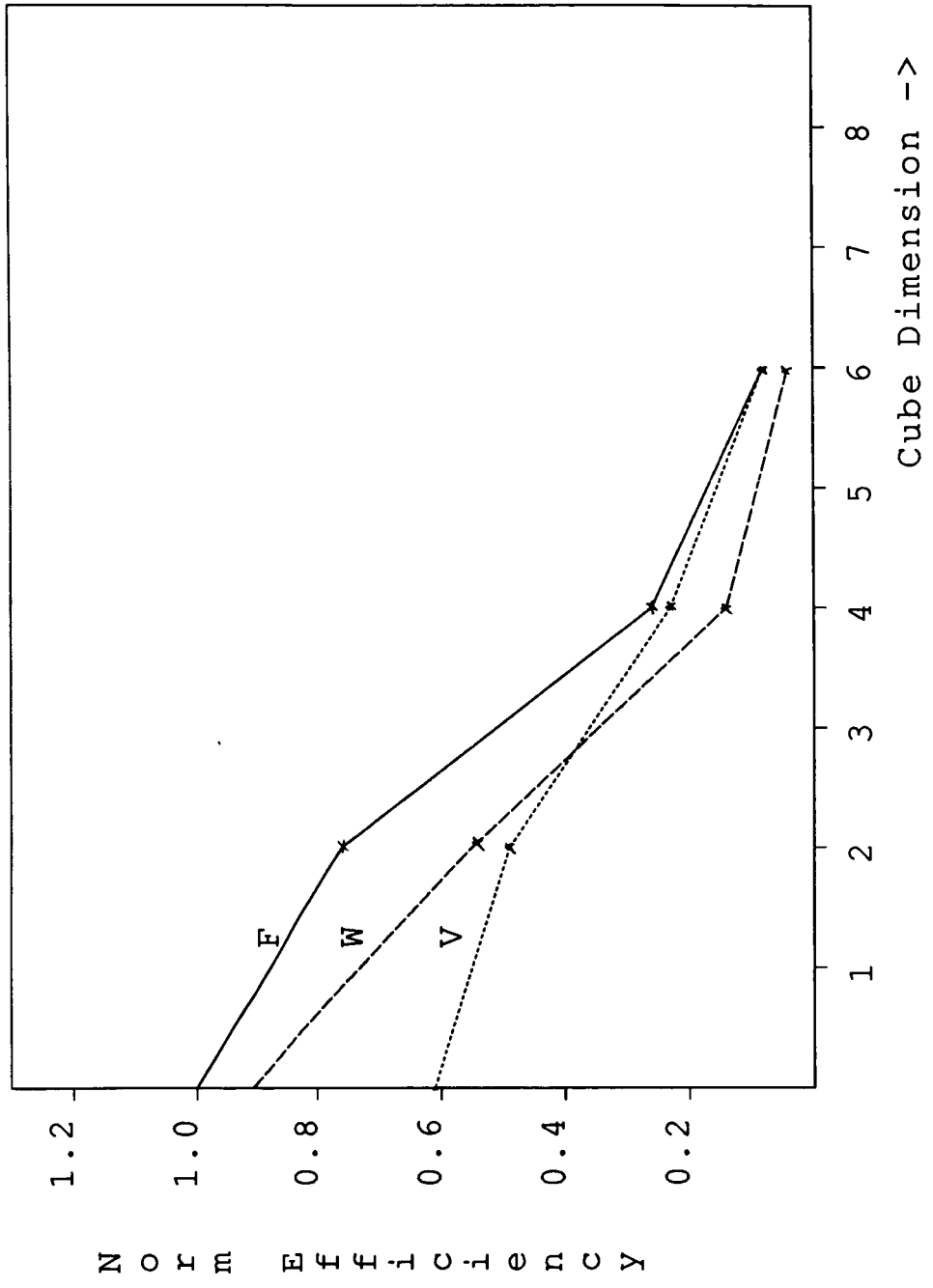


Fig. 4: Effect of Problem Size on Efficiency
Square Partitions F Cycle

Fig. 5: Normalized Efficiency vs. Cube Dimension
Square Partitions 128x128 Domain Size



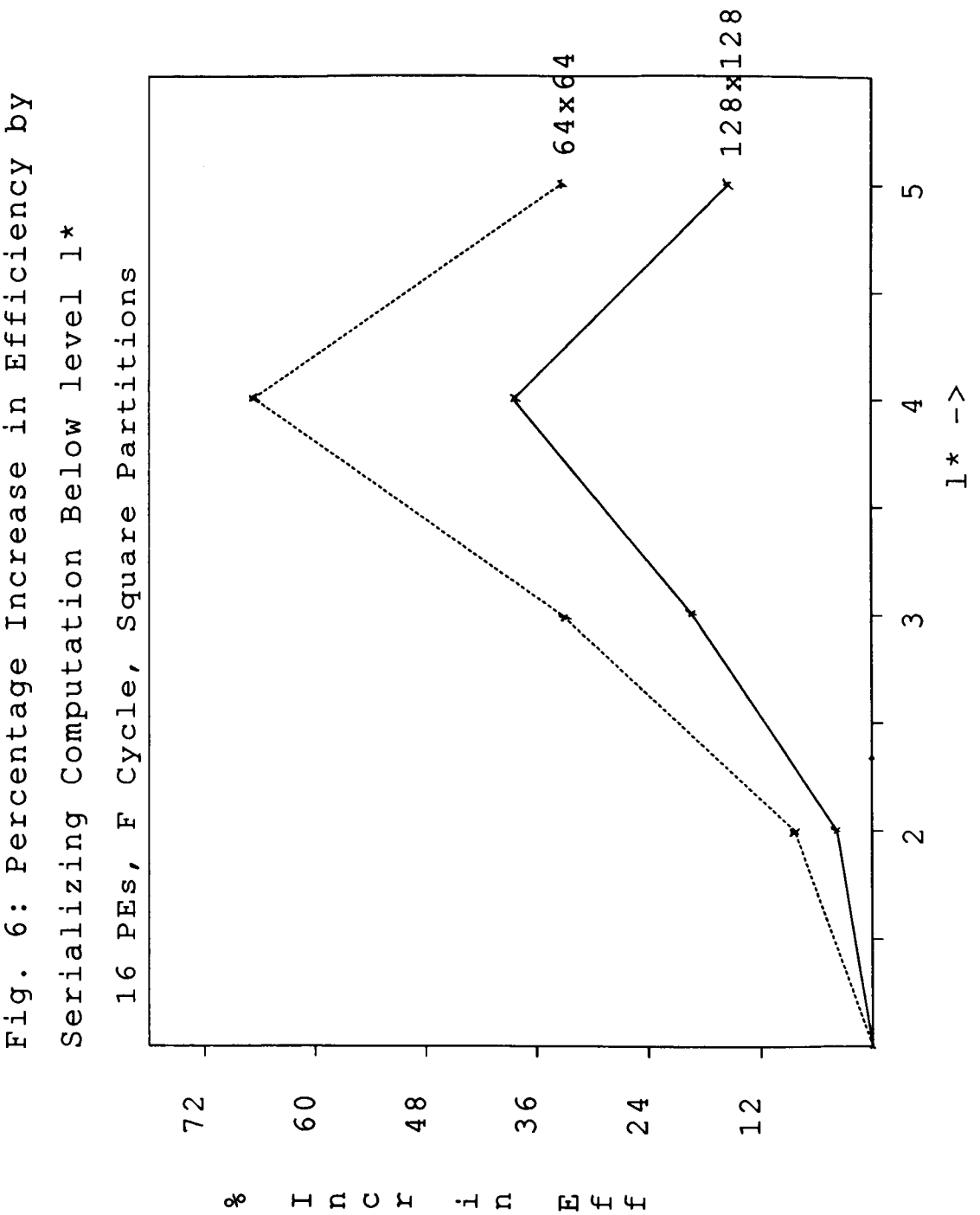


Fig. 6: Percentage Increase in Efficiency by Serializing Computation Below level l*

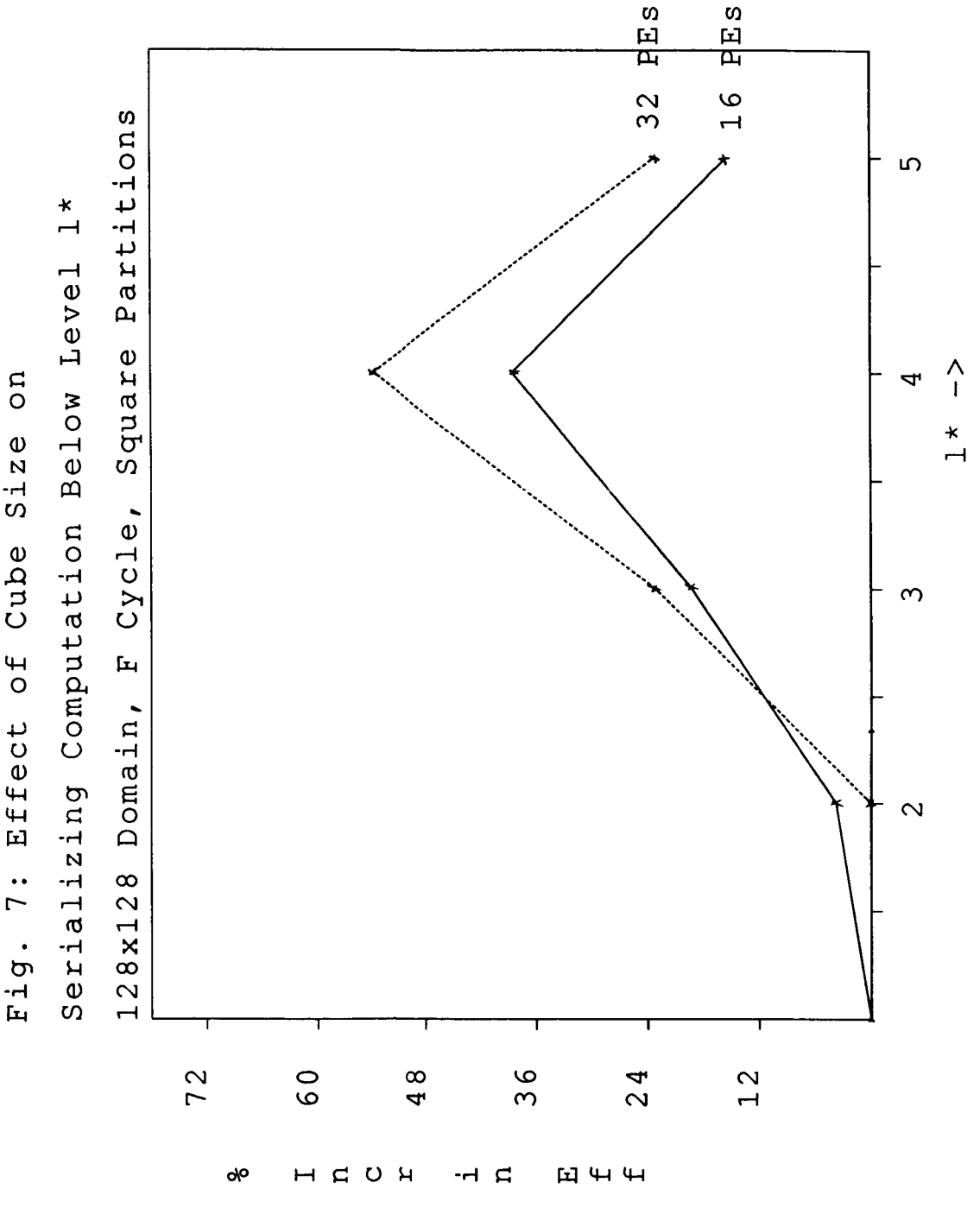


Fig. 7: Effect of Cube Size on
Serializing Computation Below Level l*
128x128 Domain, F Cycle, Square Partitions
% Incr in Eff
72
60
48
36
24
12
2
3
4
5
l* ->
32 PEs
16 PEs

Standard Bibliographic Page

1. Report No. NASA CR-178319 ICASE Report No. 87-37	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle IMPLEMENTATION OF MULTIGRID METHODS FOR SOLVING NAVIER-STOKES EQUATIONS ON A MULTIPROCESSOR SYSTEM		5. Report Date June 1987 6. Performing Organization Code
7. Author(s) Vijay K. Naik and Shlomo Ta'asan		8. Performing Organization Report No. 87-37
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		10. Work Unit No. 505-90-21-01 11. Contract or Grant No. NAS1-18107
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, D.C. 20546		13. Type of Report and Period Covered Contractor Report 14. Sponsoring Agency Code

15. Supplementary Notes

Langley Technical Monitor:
J. C. South

Final Report

Submitted to the Proc. of the 6th IMACS Int. Symp. on Computer Methods for PDE's, Lehigh Univ., PA, June 23-26, 1987

16. Abstract

In this paper we present schemes for implementing multigrid algorithms on message based MIMD multiprocessor systems. To address the various issues involved, a nontrivial problem of solving the 2-D incompressible Navier-Stokes equations is considered as the model problem. Three different multigrid algorithms are considered. Results from implementing these algorithms on an Intel iPSC are presented.

17. Key Words (Suggested by Authors(s))	18. Distribution Statement
multiprocessor systems, hypercube, multigrid algorithms, Navier-Stokes equation	62 - Computer Systems 64 - Numerical Analysis Unclassified - unlimited

19. Security Classif.(of this report)	20. Security Classif.(of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	33	A03

For sale by the National Technical Information Service, Springfield, Virginia 22161