NASA Contractor Report 178237

ICASE REPORT NO. 87-5

ICASE

PERFORMANCE STUDIES OF THE MULTIGRID ALGORITHMS IMPLEMENTED ON HYPERCUBE MULTIPROCESSOR SYSTEMS

Vijay K. Naik

Shlomo Ta'asan

Contract No. NAS1-18107

January 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

Performance Studies of the Multigrid Algorithms Implemented on Hypercube Multiprocessor Systems

Vijay K. Naik

Shlomo Ta'asan

Institute for Computer Applications in Science and Engineering

ABSTRACT

In this paper we analyze and compare the performance on a hypercube multiprocessor of some of the major multigrid techniques used in practice. The model problem considered here is that of solving the 2-D incompressible Navier-Stokes equations representing the flow between two parallel plates. Results obtained by implementing the different multigrid schemes on an iPSC are presented. Effects on the overall performance of various parameters of the algorithms, of the partitioning strategies employed, and of some of the characteristics of the underlying architecture are discussed.

Research was 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. Recently there has been increased interest in parallelizing these algorithms. Although the multigrid methods exhibit a high degree of parallelism in the individual operations involved, it is not necessarily true that these algorithms perform optimally on multiprocessor systems as well. The performance of a multiprocessor system solving a given problem depends on several parameters. These include architecture dependent parameters, algorithm dependent parameters, and implementation dependent parameters. In this paper we discuss some of the performance issues involved in the parallel implementation of these algorithms and present some experimental results obtained by solving the 2-D Navier-Stokes equations for incompressible fluid flow on the Intel's Personal SuperComputer (iPSC). All the experimental results presented here are obtained with the Release 3.0 iPSC operating system. 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 performance issues involved are discussed and some experimental results are presented. Conclusions are given in Section 5.

2. Multigrid Algorithms

Consider a differential equation given by $\mathrm{L}U = F$ with boundary conditions $BU = g$ defined on an n-dimensional domain in R^n. For simplicity of exposition let L be an elliptic operator. Let the difference scheme $\mathrm{L}^h U^h = F^h$ with boundary condition $B^h U^h = g^h$, approximate this differential equation.

Now suppose we are solving the difference scheme by relaxation (Gauss-Seidel in lexicographic order, for instance). The error here can be written as $e_n^h = U^h - u_n^h$ 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 above 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.

A closer study reveals 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 $2h$. Applying this idea recursively one arrives at a multigrid algorithm. It involves relaxation sweeps on all levels, transfer of residuals from a fine to coarse level, and interpolation of correction from coarse to fine level. An important property of such an algorithm is that the rate of convergence remains independent of the size of the problem if the order and the frequency with which the grids are visited are chosen properly.

The recursive idea described above for reducing the error in the solution of the problem gives rise to a cyclic order of computation and these 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 cycle called F cycle which is less well known. A scheme called Full MultiGrid (FMG), in which the first approximation on the fine level is obtained

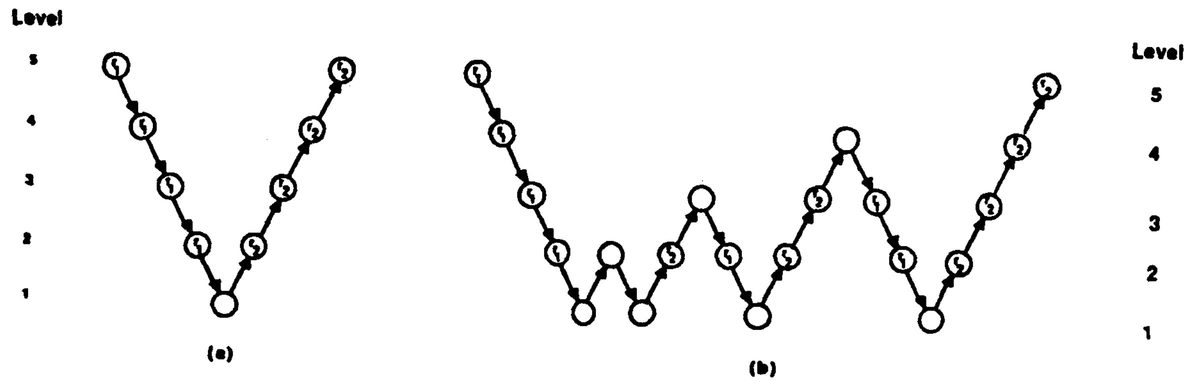


Fig. 1 (a) V Cycle (b) F Cycle

by solving a similar problem on a coarser level, yields optimal performance. A detailed description of the various multigrid techniques and the algorithms based on these cycles is given in [Brandt 1984]. For the sake of clarity, we will refer to the algorithms based on these three cycles as the V, F, and W algorithms. All three algorithms make use of the FMG scheme. Furthermore, the basic multigrid operations in these algorithms remain the same, but the number of times a given grid is visited within a multigrid cycle is different. We will see that the relative performance of these three algorithms on multiprocessor systems is not the same as on the sequential machines.

From the performance point of view the parameters that characterize these algorithms are the amount of computational work done per cycle, the number of cycles required for achieving the desired accuracy, and the order and the relative frequency with which different grids are visited. The last parameter is not so important for the sequential implementations but is a crucial factor for parallel applications.

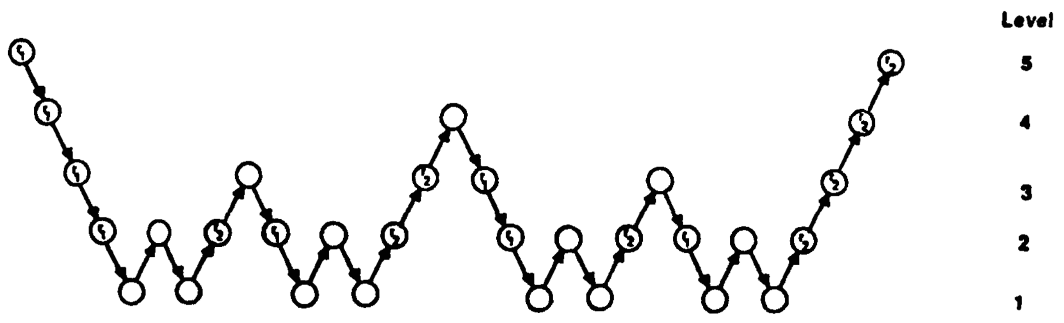


Fig. 2 W Cycle

If w_L denotes the total amount of computational work on the highest level L then it can be shown that the total work per FMG cycle for the V algorithm is asymptotically less than or equal to $\frac{16}{9} \cdot w_L$. It is less than or equal to $\frac{64}{27} \cdot w_L$ and $4 \cdot w_L$ for the F and W algorithms, respectively. These calculations assume that the number of relaxation sweeps on each grid is a small constant. If the number of multigrid cycles necessary to achieve the desired accuracy is also a small constant then all three algorithms perform optimally, i.e., in time $O(w_L)$, on sequential machines.

Characterizing the convergence properties of these algorithms is difficult, but experimental results suggest that in general the convergence rates per cycle for the W algorithm are the best and those of the V algorithm are mediocre. The F algorithm is somewhere between the two, usually slightly worse than the W algorithm. So the W algorithm is almost always preferred over the other two on sequential machines.

Figures 1 and 2 illustrate the order in which the three algorithms visit the different levels within a cycle. Here the increasing level numbers indicate finer mesh sizes. The letters within circles denote the number of relaxation sweeps on the corresponding levels. 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. The significance of this property will become clearer in Section 4 where we discuss performance issues.

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.

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

$$\begin{aligned} \Delta\psi &= \omega \\ u\ \omega_x + v\ \omega_y &= \frac{1}{\mathrm{Re}}\ \Delta\omega. \end{aligned}$$

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,

$$u = -\psi_y \text{ and } v = \psi_x .$$

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

$$\begin{array}{ll} u = v = 0 & \text{at } y = 0 \\ u = U_0, \quad v = 0 & \text{at } y = 1 \end{array}$$

Periodicity is imposed in the X direction and
a constant pressure gradient is imposed on the flow.

The details of discretizing and solving these equations using the multigrid methods on the hypercube multiprocessor system are given elsewhere[1].

4. Performance Issues in the Parallel Implementation

There are several parameters that affect the performance of a multiprocessor system employed to solve a given problem. These parameters are algorithm and implementation dependent as well as architecture dependent. One must take into account all these parameters before making a decision about the suitability of a particular algorithm for solving a problem on a multiprocessor system. In the following, we describe the interaction of some of these parameters.

4.1 Partitioning Scheme

First we describe the effect of partitioning the domain on the distribution of the computational load. For the sake of simplicity consider a 2-D square domain with

[1]V. K. Naik and S. Ta'asan, *A methodology for implementing multigrid methods in solving Navier-Stokes equations on a hypercube multiprocessor system*, in preparation.

$2^L \times 2^L$ points on the highest level L. 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. We map the partitions onto the hypercube nodes in a one-to-one fashion using the binary reflected grey code scheme [Chan 1986]. We further assume that a *fixed region* partitioning strategy is used, i.e., on the successive coarser levels each partition contains regions of the domain, formed by the points that are the coarse level counterparts of the fine level points of the partition. Under this partitioning scheme, in moving from level L to level $\max(x, y)$ the number of points associated with each partition decreases by a factor of four. 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.

With the above described properties of the partitioning scheme, it is possible to make some predictions about the performance of the various multigrid algorithms under some assumptions about the communication costs. Specifically, it is possible to develop some analytical bounds on the speedups or efficiency of the system with a given set of communication parameters. Here we consider some simple cases and present some experimental results. Detailed analytical results are presented elsewhere[1].

Consider the case where there are as many partitions as there are points on the finest level i.e., each processor is assigned one point of the domain. In the above notation this means that $x = y = L$. Assume that there are no communication costs and so the evaluated performance parameters will represent the upper bounds. We consider speedup or efficiency of the system as a measure of the performance. We define the speedup of a system with N processors as the ratio of the time taken by a single processor to solve a problem to the time taken by N processors to solve the same problem using the same algorithm. The efficiency of this system is obtained by dividing the speedup by N.

The total computational cost incurred with N processors is bounded by the computational cost of the processor that performs work on all the levels. Now for the case considered here the work on any level is equal to the work associated with a single point and so, the total computational cost C_N of the system is given by,

$$C_N = \sum_{i=1}^{L} V_i \cdot w_i$$

where, V_i is the total number of visits to level i and w_i is the maximum work per processor on level i - a constant in this case. Thus,

$$\textbf{speedup} = \frac{c \cdot N}{\sum_{i=1}^{L} V_i}$$

[1] V. K. Naik and S. Ta'asan, *Performance studies of multigrid algorithms implemented on message passing architectures*, in preparation.

$$\text{where, } c = \begin{cases} \frac{16}{9} & \text{for V Algorithm} \\ \frac{64}{27} & \text{for F Algorithm} \\ 4 & \text{for W Algorithm} \end{cases}$$

Note that here $N = 4^L$. Depending on the type of algorithm chosen, the sum over total number of visits varies. It can be shown that for the V algorithm this sum is $O(L^2)$. It is $O(L^3)$ and $O(2^L)$ for the F and W algorithms, respectively. Thus, the maximum speedup with N processors for the V algorithm is $O\left(\frac{N}{\log_4^2 N}\right)$, for the F algorithm it is $O\left(\frac{N}{\log_4^3 N}\right)$, and for the W algorithm it is $O(N^{\frac{1}{2}})$. This shows that when there are as many processors as there are number of points on the fine level, the speedup for the W algorithm is far from being optimal even when the communication costs are ignored.

For the cases where the number of points assigned to each partition on the highest grid is more than one, the expressions for the speedups are more complex. In general the bounds improve. Here we present some experimental results. The effects of the algorithm dependent properties and of the partition size on the computational efficiency are shown in Fig. 3. The results shown in this figure are obtained by measuring only the computational costs on the iPSC. The efficiencies for each algorithm are normalized with respect to the computational cost of solving the problem using that algorithm on a uniprocessor. Thus, although the efficiencies of the three algorithms shown in Fig. 3 cannot be compared directly against one another, these curves provide information about the relative degradation in performance as more processors are added or as the partition size is decreased and these trends can be compared. Clearly, the sensitivity to the partition size is different for the three algorithms; V algorithm is the least sensitive and W algorithm is the most sensitive. Another point to be noted is that for the smaller size hypercubes i.e., when the partition size is large, all three algorithms show improvements in efficiency and the difference in the efficiencies of the three algorithms decreases. The effect of the size of the partitions is shown explicitly in Fig. 4 for the F algorithm. The other two algorithms show similar trends. As in Fig. 3 the measurements are made without including the communication costs. The results presented in these two figures suggest that even if communication is instantaneous, the attainable speedup or efficiency is low if the amount of work per partition is small. If the communication costs are included in the measurements then the performance deteriorates as shown by Fig. 5 for the F algorithm.

The efficiency discussed above represents a measure of the ability of an algorithm to keep the processors of the system busy. It does not include the effect of the numerical properties of the algorithm. If one is interested in the minimum overall cost of solving the problem, then both of these properties must be taken into account. The numerical properties are usually dependent on the problem being considered and so cannot be characterized easily. For the V algorithm these properties sometimes depend on the mesh size also. For the problem we are considering here both the F and W algorithms need about two FMG cycles to solve a 128 x 128 problem to the level of discretization error, whereas the V algorithm takes about seven cycles for the same. To compare the three algorithms more accurately we compute the efficiencies using the best sequential timings which for this problem are given by the F algorithm. We refer to such an efficiency as the normalized efficiency. In all cases the problem is solved to get the same level of numerical accuracy. These results are shown in Fig. 6. Note that here the communication costs are included in computing the normalized efficiency. It can be seen that when the partition sizes are large and the hypercube size is small, both the F and W algorithms perform better than the V algorithm in spite of the adverse communication costs. For small partition sizes the V algorithm may perform better even though its convergence properties are inferior.

4.2 Partition Shape

When the number of points per partition on the finest level is more than one, the shape of the partitions is another parameter that has to be taken into account. Reed et al. [Reed 1986] have discussed in detail the combined effect of the iteration stencil, the partition shape, and the communication parameters of the underlying architecture on the total communication cost. Their discussion concentrates on minimizing the communication cost assuming that the computational work is evenly distributed and remains the same through out 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 we use square partitions, 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.

Experimental results showing the efficiency of partitions with different shapes in balancing the computational load are shown in Fig. 7. Note that the communication costs are not taken into account. Here a domain with 64x64 points on the fine level is subdivided into 64 partitions. The four different cases considered consisted of partitions with 8x8 points (8 points in x direction and 8 in y direction), 16x4 points, 32x2 points, and 64x1 points on the fine level. In the first case we get square partitions whereas in the last case we get strips. As expected the squares balance the computational load better than any other shapes considered. The results shown here correspond to the F algorithm. For the W algorithm these will be more pronounced, but less so for the V algorithm.

When communication costs are introduced the shape of the partitions may affect the performance differently. For the iPSC the cost of initializing a message is orders of magnitude higher than sending a single byte across a channel. A single packet can contain up to 1024 bytes. In addition, all the channels leaving or entering a node cannot be effectively utilized for simultaneously sending or receiving messages. For the problem sizes we have considered here, it turns out that the communication costs on the iPSC for the strips are less than those for the squares. This is shown in Fig. 8 for the F algorithm applied to a problem with 64x64 points on the highest level. Here the efficiency is based on the computation plus communication cost. It is obvious from Figures 7 and 8, that for the problem sizes we are considering, the communication costs incurred with strips are much less than those for the squares. In general this is not true. The problem sizes considered here are special cases and it can be easily shown that for strips, asymptotically, the total number of packets for the two messages sent out per exchange of information with the neighboring partitions is greater by a constant factor than those for the four messages sent out by the square partitions when the number of interior points per partition is the same [Reed 1986].

4.3 Schemes for Reducing the Communication Costs

In the partitioning schemes 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 partion 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.

Experimental results showing the performance improvements brought about by serializing the work below some level l^* by moving all the regions to a single node, are shown in Fig. 9. Here the percent increase in efficiency by serializing the work below levels 2 through 5 on a 16 PE hypercube is shown for the two problems with the finest level domain having 64x64 and 128x128 points. In this figure $l^* = 1$ corresponds to the fixed region partitioning scheme i.e., no moving takes place. It can be seen that the performance peeks out at a particular value of l^*. The savings in 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 bigger size problem. In Fig. 10 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.

5. Conclusions

We have considered the performance issues involved in implementing the multigrid methods on a hypercube multiprocessor system. It is shown that both algorithm dependent as well as implementation dependent parameters affect the performance considerably and the selection of an algorithm or of a partioning scheme must be based on the combined effect of these parameters. We demonstrate by some experimental and analytical results that the best sequential algorithm may not always be the most suitable algorithm for parallel processing. At the same time an algorithm that gives the best speedups may not be the most suitable candidate either. By using the problem of solving the 2-D Navier-Stokes equations as a model problem we show that a less well known method given by the F algorithm gives the best performance. We have also shown that when the communication costs are high instead of balancing the computational load it may be advantageous to sequentialize some parts of the work and avoid communication costs in those sections.

ACKNOWLEDGEMENT

We would like to thank the Mathematics and Statistics Research Section at Oak Ridge National Laboratory for generously granting us the use of their iPSC. We are particularly indebted to Tom Dunigan for his assistance during the course of this work. We are grateful to Bob Voigt for constant support and encouragement and to Merrell Patrick for improving the style of the paper. Thanks to Amr Fahmy for his time and and help in preparing the figures.

REFERENCES

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

(2) T. F. Chan and Y. Saad, *Multigrid algorithms on the hypercube multiprocessor*, IEEE Trans. Comput., 35 (1986), pp. 969-977.

(3) D. A. Reed, L. M. Adams, M. L. Patrick, *Stencils and problem partitionings: Their influence on the performance of multiple processor systems*, IEEE Trans. Compt., to appear.

Fig. 3: Effect of Algorithm Characteristics
on Computational Efficiency
Square Partitions, 128x128 Domain

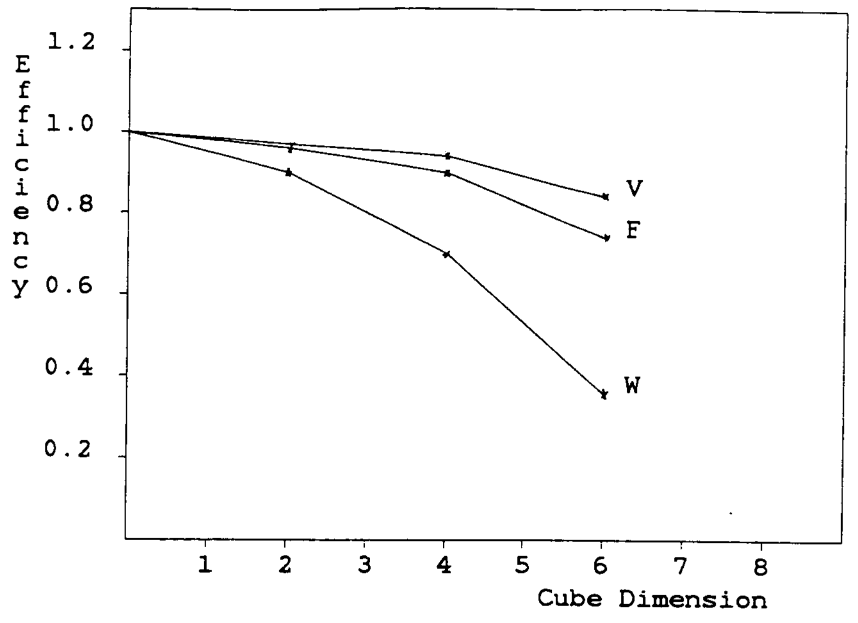


Fig. 4: Effect of Partition Size
on Computational Efficiency
Square Partitions, F Cycle

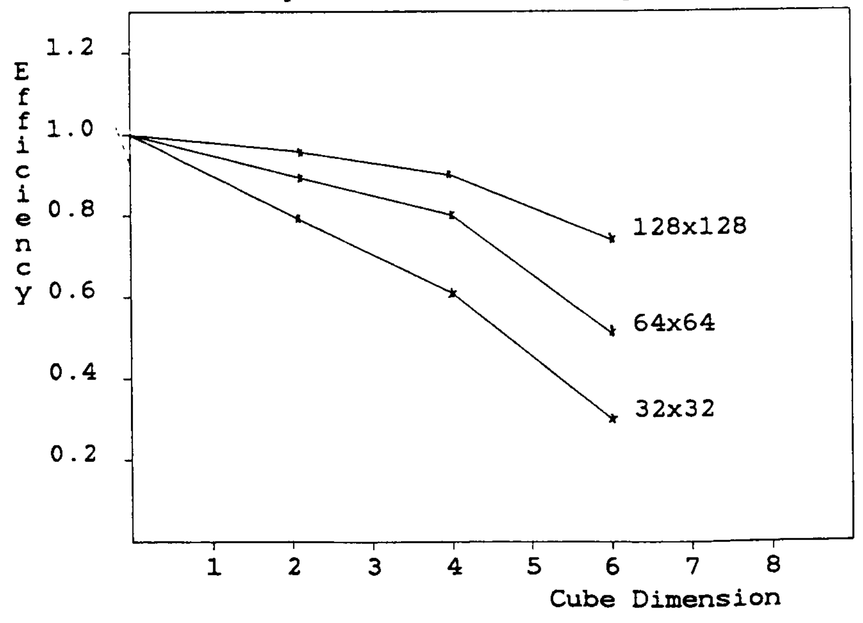


Fig. 5: Efficiency vs. Cube Dimension
Square Partitions, F Cycle

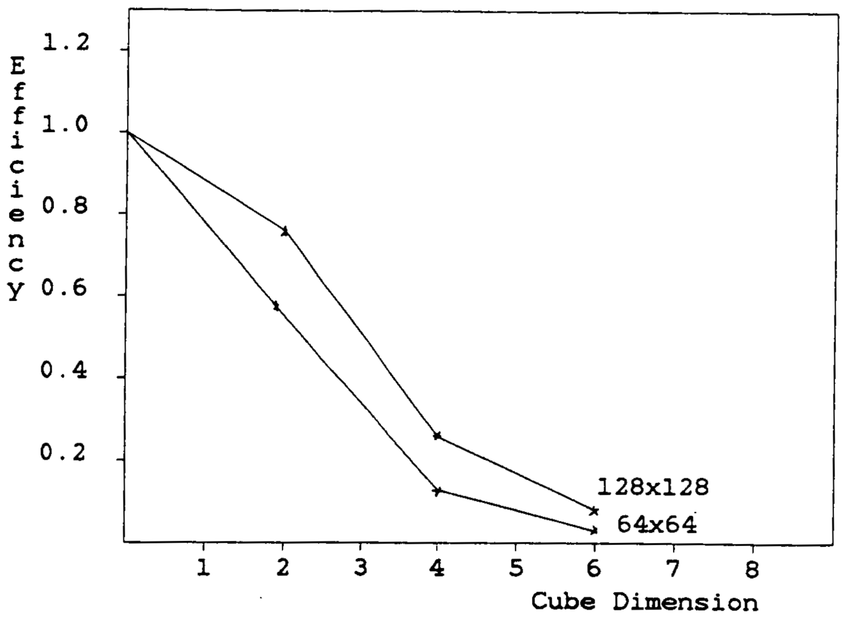


Fig. 6: Effect of Normalizing Efficiency
(Communication Costs Included)
Square Partitions, 128x128 Domain

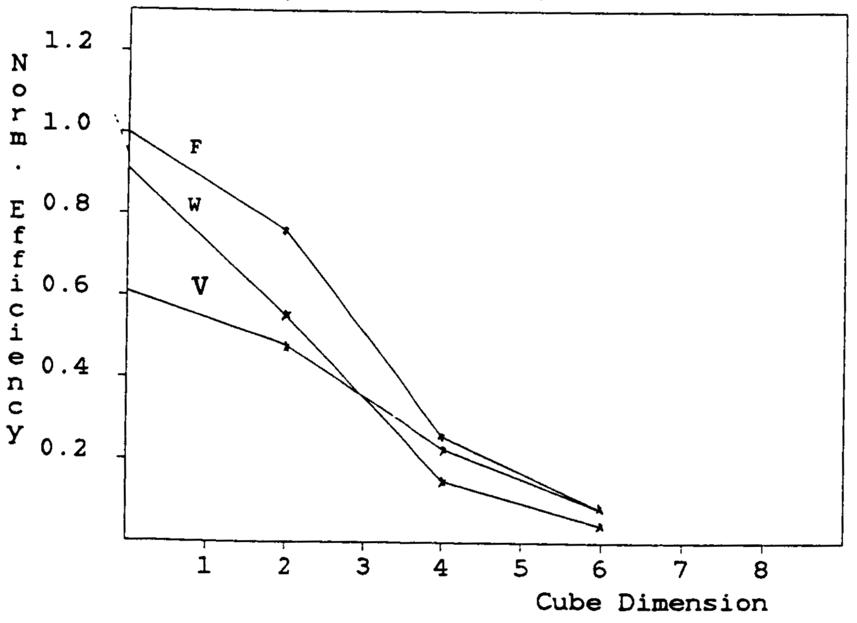


Fig. 7: Effect of Partition Shape on Computational Efficiency
64x64 Domain, 64 PEs, F Cycle

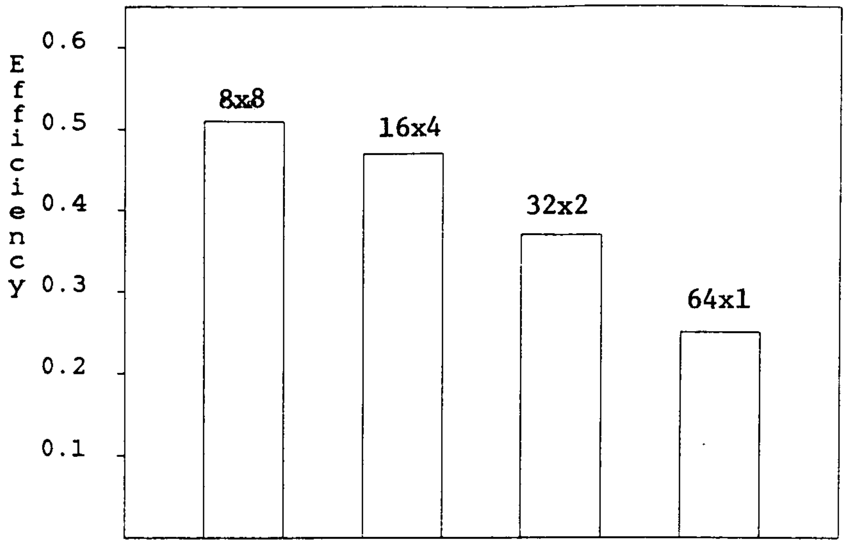


Fig. 8: Effect of Partition Shape on Efficiency (Communication Costs Inc.)
64x64 Domain F Cycle

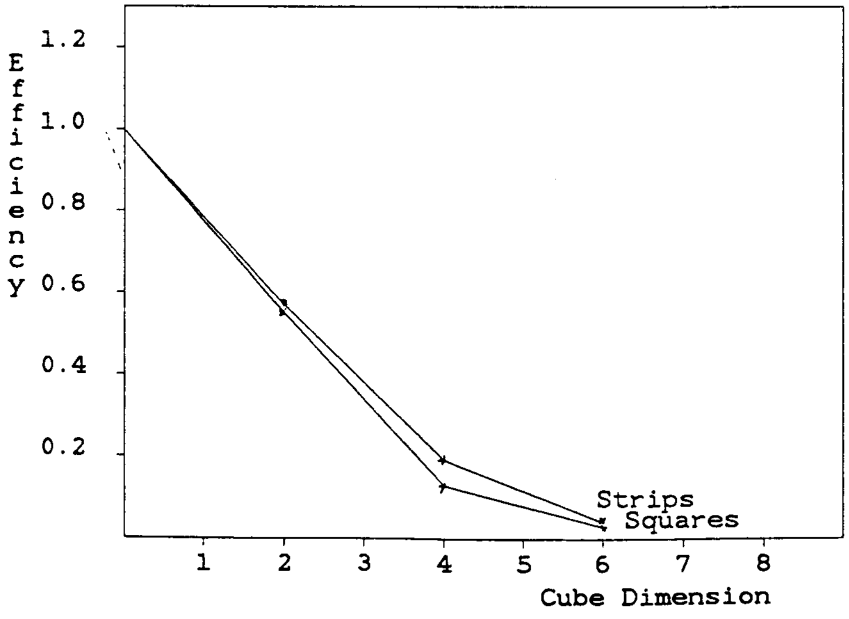


Fig. 9: Percent Change in Efficiency by Serializing Computation Below Level l*
Square Partitions, F Cycle, 16 PEs

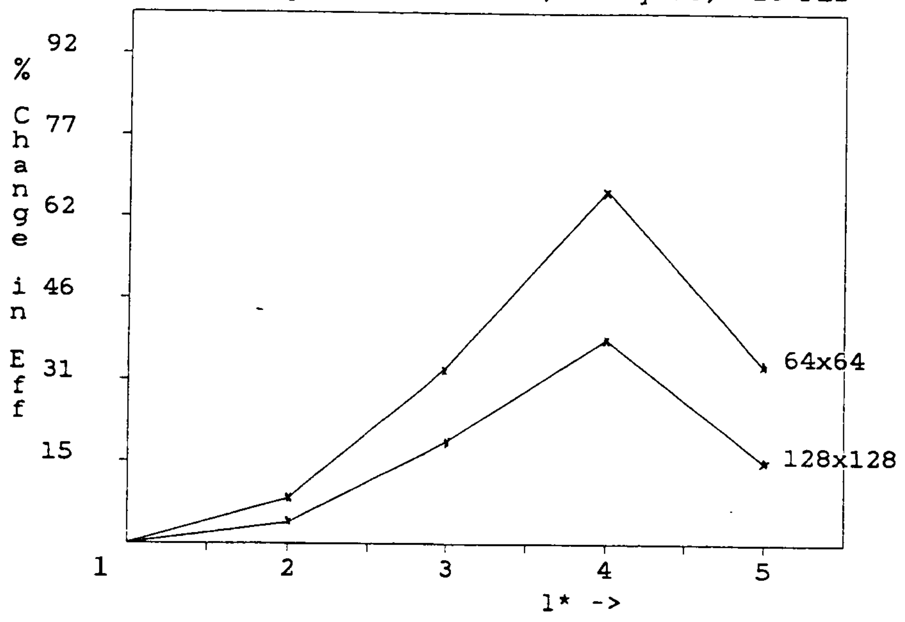
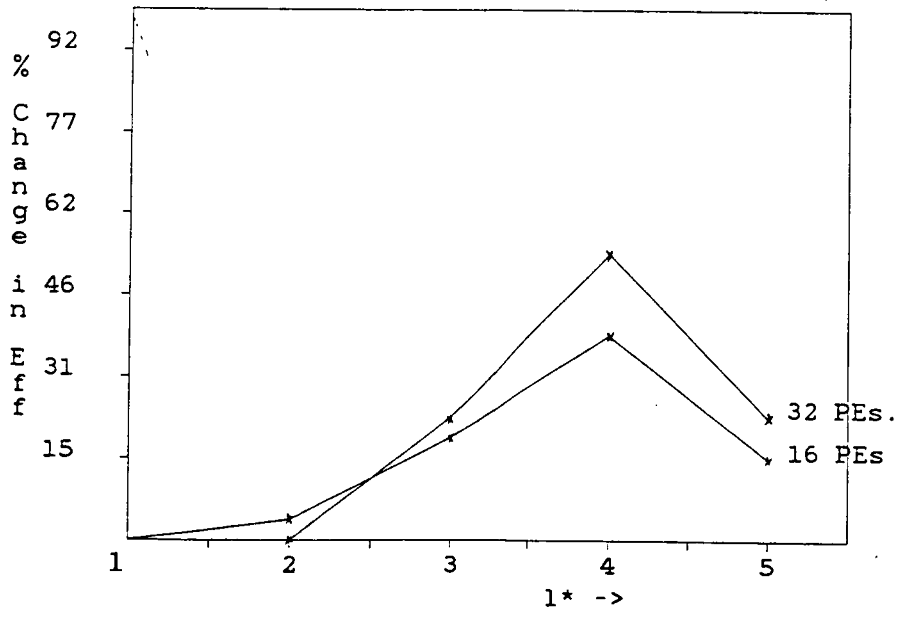


Fig. 10: Effect of Cube Size on Serializing Computation Below Level l*
Square Partitions, 128x128 Domain, F Cycle



Standard Bibliographic Page

1. Report No. NASA CR-178237 ICASE Report No. 87-5	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle PERFORMANCE STUDIES OF THE MULTIGRID ALGORITHMS IMPLEMENTED ON HYPERCUBE MULTIPROCESSOR SYSTEMS		5. Report Date January 1987
		6. Performing Organization Code
7. Author(s) Vijay K. Naik and Shlomo Ta'asan		8. Performing Organization Report No. 87-5
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.
		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 ~~505-90-21-01~~

15. Supplementary Notes

Langley Technical Monitor:
J. C. South

Submitted to Proc. of the 2nd Hypercube Multiprocessor Conference

Final Report

16. Abstract

In this paper, we analyze and compare the performance on a hypercube multiprocessor of some of the major multigrid techniques used in practice. The model problem considered here is that of solving the 2-D incompressible Navier-Stokes equations representing the flow between two parallel plates. Results obtained by implementing the different multigrid schemes on an iPSC are presented. Effects on the overall performance of various parameters of the algorithms, of the partitioning strategies employed, and of some of the characteristics of the underlying architecture are discussed.

17. Key Words (Suggested by Authors(s)) performance, multigrid algorithms, hypercube multiprocessors, Navier-Stokes equations		18. Distribution Statement 62 - Computer Systems 64 - Numerical Analysis Unclassified - unlimited	
19. Security Classif.(of this report) Unclassified	20. Security Classif.(of this page) Unclassified	21. No. of Pages 21	22. Price A02

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

NASA Langley Form 63 (June 1985)