CONF-890844--1

PERFORMANCE OF A PARALLEL ALGORITHM FOR SOLVING THE NEUTRON DIFFUSION EQUATION ON THE HYPERCUBE*

Bernadette L. Kirk
Radiation Shielding Information Center
Yousry Y. Azmy
Analysis and Design of Advanced Systems
Engineering Physics and Mathematics Division
Oak Ridge National Laboratory
P.O. Box 2008
Oak Ridge, Tennessee 37831-6362
Phone 615-574-6176
Telex(Answer Back): 854467(ORNL EPIC UD)

CONF-890844--1

DE89 006128

Abstract

The one-group, steady state neutron diffusion equation in two-dimensional Cartesian geometry is solved using the nodal method technique. By decoupling sets of equations representing the neutron current continuity along the length of rows or columns of computational cells a new iterative algorithm is derived that is more suitable to solving large practical problems. This algorithm is highly parallelizable and is implemented on the Intel iPSC/2 hypercube in three versions which differ essentially in the total size of communicated data. Even though speedup was achieved, the efficiency is very low when many processors are used leading to the conclusion that the hypercube is not as well suited for this algorithm as shared memory machines.

KEYWORDS: PARALLEL-ALGORITHM; HYPERCUBE; DIFFUSION; NODAL; PERFORMANCE

*Research sponsored by the U.S. Department of Energy Office of Magnetic Fusion under Contract No. DE-AC05-84OR21400 with Martin Marietta Energy Systems, Inc.

PERFORMANCE OF A PARALLEL ALGORITHM FOR SOLVING THE NEUTRON DIFFUSION EQUATION ON THE HYPERCUBE

Bernadette L. Kirk and Yousry Y. Azmy

Abstract

The one-group, steady state neutron diffusion equation in two-dimensional Cartesian geometry is solved using the nodal method technique. By decoupling sets of equations representing the neutron current continuity along the length of rows or columns of computational cells a new iterative algorithm is derived that is more suitable to solving large practical problems. This algorithm is highly parallelizable and is implemented on the Intel iPSC/2 hypercube in three versions which differ essentially in the total size of communicated data. Even though speedup was achieved, the efficiency is very low when many processors are used leading to the conclusion that the hypercube is not as well suited for this algorithm as shared memory machines.

1. INTRODUCTION

Elliptic partial differential equations (PDEs) abound in many scientific disciplines. In reactor physics, for example, one form in which they appear is the neutron diffusion equation. The standard approach in finding a good approximate solution to an elliptic equation is by subdividing the coordinate systems into intervals, thereby producing a finite mesh. Discrete variable equations are then derived from the PDEs, and solved for the dependent discrete-variables, locally defined with respect to the mesh. Finite-difference methods require very fine meshes to model the neutron flux accurately. Translated into a computing environment, these methods require large memory and long CPU time. Nodal methods, on the other hand, have been shown to produce good approximate solutions with the use of coarse meshes that results in a high computational efficiency (Lawrence, 1985). Recent development of parallel computing architectures have introduced a new challenge for numerical analysts; namely the development of algorithms used to solve the discrete-variable equations in parallel so that high speedups are achieved. Indeed elliptic PDEs have been solved on the hypercube architecture (Chan et al., 1986; Haghoo and Proskurowski, 1988). Also parallel algorithms were developed for solving the

neutron diffusion equation on shared memory machines (Zee and Turinsky, 1987; Rajic and Ougouag, 1988; Kirk and Azmy, 1989).

In this paper we describe a new algorithm for solving the nodal method diffusion equations which is particularly suitable for parallelization. The algorithm is based on decoupling the discrete-variable equations into three sets: two tridiagonal systems representing the neutron current continuity in the x and y directions for each row and column, respectively, and a five-point scheme system representing the balance of neutrons over computational cells. This feature is unique to the nodal method equations, so that the results and conclusions reached in this work are valid only for this method.

A major concern in developing algorithms for parallel machines is the adequacy of the parallel architecture to produce significant improvement in the performance of the algorithm as the number of processors participating in the solution is increased. We present three variations of our new algorithm implemented on a distributed memory machine, INTEL's Hypercube. Extensive numerical testing of these variants show that speedups are obtainable on the hypercube, but that exploiting its full potential cannot be achieved even when extremely large problems are solved. Comparison of the performance of the three programming schemes reveals the crucial effect of the size of communicated messages on the speedup and efficiency.

In Section 2 we briefly outline the derivation of the nodal diffusion method equations, and we discuss the two methods used to solve them, namely the direct and iterative methods. In Section 3 we describe the successive relaxation acceleration method that we applied to the iterative method, and we present results to demonstrate its effectiveness in reducing the number of iterations significantly. In Section 4 we present the parallel algorithm used to solve the nodal diffusion equations on the hypercube. Three variants of this algorithm that differ from one another in the total size of communicated messages are compared in Section 5. The adequacy of parallel architectures of the message passing type to solving the nodal diffusion method equations using the algorithm presented here is discussed in Section 6. Finally, our conclusions are summarized in Section 7.

2. THE NODAL METHOD FOR THE NEUTRON DIFFUSION EQUATION

In two-dimensional cartesian geometry, the one-group steady-state neutron diffusion equation is:

$$D\left[\frac{\partial^2\phi}{\partial x^2}+\frac{\partial^2\phi}{\partial y^2}\right]-\sigma\phi=-S\ , \tag{1}$$

where D is the diffusion coefficient, σ is the macroscopic removal cross section, ϕ is the neutron scalar flux, and S is the volumetric external source of neutrons.

By dividing the problem domain into two-dimensional closed intervals of the form $[-a_m,+a_m]\times[-b_m,+b_m]$, respectively, for $m=1,...,L$, a nodal method solution can be derived locally. Since details of the derivation are presented elsewhere (Kirk and Azmy, 1989), only an outline of the development of the nodal method neutron diffusion equation is included here.

First we average Eq. (1) over a computational cell to obtain a balance equation relating the cell-averaged flux to the neutron current on the four surfaces bounding that cell. Next by transverse-averaging Eq. (1) with respect to x and y separately we obtain two ODEs with the transverse-averaged flux as dependent variable. These can be solved by approximating the leakage term in a truncated expansion, resulting in two expressions for the transverse-averaged flux within the computational cell in terms of its value on the cell edges. These expressions are used to derive additional relations between the cell-averaged flux and the cell-surface currents. Finally, the cell-surface currents are eliminated from the neutron balance equation yielding,

$$\begin{aligned}\frac{D_m}{2}\Bigg\{&\left(\frac{\gamma_m^2P_m^x}{1-P_m^x}\right)\left(\bar{\phi}^y_{+m}-2\bar{\bar{\phi}}_m+\bar{\phi}^y_{-m}\right)+\left(\frac{\gamma_m^2P_m^y}{1-P_m^y}\right)\\&\left(\bar{\phi}^x_{+m}-2\bar{\bar{\phi}}_m+\bar{\phi}^x_{-m}\right)\Bigg\}-\gamma_m\bar{\bar{\phi}}_m=-S_m\ ,\quad m=1,...,L,\quad ,\end{aligned} \tag{2}$$

where $\gamma_m^2\equiv\sigma_m/D_m$, σ_m and D_m being the average values of σ and D over cell m, $P_m^x\equiv\tanh(\gamma_m a_m)/\gamma_m a_m$, and $P_m^y\equiv\tanh(\gamma_m a_m)/\gamma_m a_m$. In Eq. (2) the cell-averaged flux is defined by,

$$\bar{\bar{\phi}}_m\equiv\frac{1}{(4a_mb_m)}\int_{-a_m}^{+a_m}dx\int_{-b_m}^{+b_m}dy\phi(x,y)\ ,$$

and the cell-averaged source, S_m, is defined analogously; while the x-averaged, surface-evaluated flux is defined by,

$$\bar{\phi}^x_{\pm m} \equiv (\frac{1}{2a_m}) \int_{-a_m}^{+a_m} dx \phi(x, \pm b_m) \ ,$$

and the y-averaged, surface-evaluated flux, $\bar{\phi}^y_{\pm m}$, is defined analogously. Clearly the set of cell-averaged and transverse-averaged (i.e., x- and y-averaged) fluxes constitute the set of discrete-variables solved for in the nodal diffusion method.

In addition to the balance equation, an equation expressing the continuity of the neutron x-current across a y =constant surface of adjacent cells is necessary,

$$\begin{aligned} \bar{\phi}^y_{+m} \left[D_m \left(\frac{1+\omega^x_m}{2a_m P^x_m} \right) + D_l \left(\frac{1+\omega^x_l}{2a_l P^x_l} \right) \right] - \bar{\phi}^y_{-m} D_m \left(\frac{1-\omega^x_m}{2a_m P^x_m} \right) \\ - \bar{\phi}^y_{+l} D_l \left(\frac{1-\omega^x_l}{2a_l P^x_l} \right) - \bar{\bar{\phi}}_m \left(\frac{D_m \omega^x_m}{a_m P^x_m} \right) - \bar{\bar{\phi}}_l \left(\frac{D_l \omega^x_l}{a_l P^x_l} \right) = 0 \ , \end{aligned} \tag{3}$$

where $w^x_m \equiv (\gamma_m a_m P^x_m)^2/(1-P^x_m)$, and the l-th computational cell is adjacent to the m-th computational cell in the positive x-direction such that the two surfaces $x = +a_m$ and $x = -a_l$ coincide. Analogously, the y-current continuity equation is:

$$\begin{aligned} \bar{\phi}^x_{+m} \left[D_m \left(\frac{1+\omega^y_m}{2b_m P^y_m} \right) + D_n \left(\frac{1+\omega^y_n}{2b_n P^y_n} \right) \right] - \bar{\phi}^x_{-m} D_m \left(\frac{1-\omega^y_m}{2b_m P^y_m} \right) \\ - \bar{\phi}^x_{+n} D_n \left(\frac{1-\omega^y_n}{2b_n P^y_n} \right) - \bar{\bar{\phi}}_m \left(\frac{D_m \omega^y_m}{b_m P^y_m} \right) - \bar{\bar{\phi}}_n \left(\frac{D_n \omega^y_n}{b_n P^y_n} \right) = 0 \ , \end{aligned} \tag{4}$$

where cell n is adjacent to cell m in the positive y-direction. Equations (2), (3), and (4) combined with the boundary conditions form a linear set of equations that has as many equations as unknowns. In the remainder of this section we explore two methods for solving this system, the direct method and the iterative method.

2.1 THE DIRECT METHOD FOR SOLVING THE SYSTEM OF EQUATIONS

Equations (2), (3), and (4) constitute a matrix equation of the form:

$$A\vec{\phi}^x + B\vec{\phi}^y + C\vec{\phi} = D \ , \tag{5}$$

where $\vec{\phi}^x$ is the vector consisting of the surface fluxes, $\bar{\phi}^x_{-1}$, $\bar{\phi}^x_{+1}$, $\bar{\phi}^x_{+2}, \ldots, \bar{\phi}^x_{+L}$; $\vec{\phi}^y$ is the vector of surface fluxes $\bar{\phi}^y_{-1}$, $\bar{\phi}^y_{+1}$, $\bar{\phi}^y_{+2}, \ldots$, $\bar{\phi}^y_{+L}$ and $\vec{\phi}$ is the vector

of cell-averaged fluxes $\bar{\bar{\phi}}_1$, $\bar{\bar{\phi}}_2, ..., \bar{\bar{\phi}}_L$. In equation (5), A, B, C are coefficient matrices and D is a one-dimensional vector containing the volumetric and boundary sources.

The direct solution of Eq. (5) is obtained by combining the terms of the left hand side into the product of one matrix and the vector made up of $\vec{\phi}^x$, $\vec{\phi}^y$, and $\vec{\phi}$, then solving the resulting system using double precision matrix equation solvers DGEFA and DGESL from the LINPACK subroutines (Dongarra, 1978).

2.2 THE ITERATIVE METHOD FOR SOLVING THE SYSTEM OF EQUATIONS

By de-coupling equation (2) from (3) and (4), an iterative procedure can be achieved. This is done in the following manner. Initial estimates of the cell-averaged fluxes, $\bar{\bar{\phi}}_m$, are given, thus reducing the current continuity equations (3) and (4) to tridiagonal systems of equations, each representing a column or a row, respectively, where the unknowns become only the surface fluxes. Each tridiagonal system constitutes an independent process and is solved separately (with the LINPACK routine DGTSL) for the surface fluxes. These newly obtained quantities are substituted into equation (2) to update the cell-averaged fluxes. The updated values of the cell-averaged fluxes are tested against those in the preceding iteration. Convergence is achieved, and the calculation is successfully terminated if the pointwise relative difference between two consecutive iterates is less than or equal to a pre-determined small value of ϵ.

2.3 COMPARISON OF THE DIRECT VERSUS THE ITERATIVE METHOD

In order to compare the performance of the direct and the iterative methods for solving the neutron diffusion nodal method equations, they were applied to a simple test problem. The measured CPU time for each of the two methods, as well as the total number of unknowns ($\sim 3L$) for a range of mesh sizes is presented in Table I. These results were obtained on a Data General sequential computer with 4 megabytes of memory.

Table I. Comparison of the performance of the direct vs iterative methods for a simple test problem with $\epsilon = 1.0E-04$

Mesh size	Number of unknowns for direct method	Direct method (seconds)	Iterative method (seconds)
8×8	176	16.52	19.26
10×10	280	54.43	45.45
12×12	408	151.29	90.74
14×14	560	367.07	163.63
16×16	736	776.97	271.68
18×18	936	1609.93	431.11
20×20	1160		632.26
22×22	1408		899.39

As Table I indicates, the number of unknowns grows rapidly with refining the mesh for the direct method. This translates to increased computational times and larger memory requirement. The iterative method outperforms the direct method in CPU time starting from mesh sizes greater than 8×8. The unavailable values in the table for the direct method for mesh sizes 20×20 and 22×22 are due to memory limitation on the size of the matrix equation that can be solved directly with 4 megabytes of memory. This constraint does not limit the iterative method for any conceivable mesh, since the matrix equations solved in this method are tridiagonal and of the order $\sqrt{L}$ only.

3. ACCELERATING THE ITERATIVE METHOD

To reduce the number of iterations and improve the convergence, successive overrelaxation is applied on the cell-averaged fluxes $\bar{\bar{\phi}}_m$. Let $\bar{\bar{\phi}}_m^i$ signify the i-th iterate of the cell-averaged flux. Then equation (2) can be written as

$$\left[D_m\left(\frac{\gamma_m^2 P_m^x}{1-P_m^x}+\frac{\gamma_m^2 P_m^y}{1-P_m^y}\right)+\sigma_m\right]\bar{\bar{\phi}}_m^i = \left[S_m+\frac{D_m}{2}\left\{\frac{\gamma_m^2 P_m^x}{1-P_m^x}(\phi_m^{y(i)}+\phi_{-m}^{y(i)})+\frac{\gamma_m^2 P_m^y}{1-P_m^y}(\phi_m^{x(i)}+\phi_{-m}^{x(i)})\right\}\right] \quad , \tag{6}$$

where $\phi_m^{y(i)}$, $\phi_{-m}^{y(i)}$, $\phi_m^{x(i)}$, $\phi_{-m}^{x(i)}$ are the i-th values of the surface fluxes obtained from the solution of the two current continuity equations.

The above equation can be expressed as

$$\bar{\bar{\phi}}_m^i = K^i \quad , \tag{7}$$

where K^i is the right hand side of Eq. (6) divided by the coefficient on the left hand side.

Multiplying both sides of Eq. (7) by a factor w we get

$$w\bar{\bar{\phi}}_m^i = wK^i \quad . \tag{8}$$

Adding $(1-w)\bar{\bar{\phi}}_m^i$ to both sides of Eq. (8), we obtain

$$(1-w)\bar{\bar{\phi}}_m^i + w\bar{\bar{\phi}}_m^i = (1-w)\bar{\bar{\phi}}_m^i + wK^i \quad , \tag{9}$$

which simplifies to

$$\bar{\bar{\phi}}_m^i = (1-w)\bar{\bar{\phi}}_m^i + wK^i \quad . \tag{10}$$

Finally, by replacing $\bar{\bar{\phi}}_m^i$ by $\bar{\bar{\phi}}_m^{i-1}$ on the right hand side of Eq. (10), we get the new iterate expressed as

$$\bar{\bar{\phi}}_m^i = (1-w)\bar{\bar{\phi}}_m^{i-1} + wK^i \quad . \tag{11}$$

Different values of w in the range $1 < w \leq 2$ were used for Eq. (11). For the algorithm the value chosen after several computer test runs is $w = 2$ (Ames, 1965). This appears to be the optimum value.

Figure 1 shows the reduction in the number of iterations. The acceleration factor is almost 2.

The results for the hypercube in the following pages reflect the accelerated iterative method.

4. APPLICATION OF THE ITERATIVE METHOD ON THE HYPERCUBE

In the iterative method, the surface flux equations (3) and (4) for each row or column of computational cells, respectively, forms a tridiagonal system of equations which is completely uncoupled to tridiagonal matrix equations corresponding to other rows or columns. Hence, each such system can be solved independently of the others. For an $n \times m$ mesh, for example, there are n x-current continuity equations involving the surface fluxes in the x-direction for each row, and m equations in the y-direction for each column. These constitute $n + m$ independent processes which can be executed simultaneously. This characteristic of the iterative method readily lends itself to a parallel solution algorithm on the hypercube and other parallel machines.

The following steps summarize the design of the algorithm on the hypercube (Intel iPSC/2). The "host" program will assign the number of processors. The "node" program will involve mainly performing the iterations by solving the tridiagonal systems of equations.

Step 1: The 'host' program gives each processor a copy of the 'node' program. It also sends initial guesses of the cell-averaged fluxes $\bar{\bar{\phi}}_m$ (set to 0 at the start) and mesh size information to the nodes. Node 0 is designated problem manager.

Step 2: Each processor is assigned a set of tridiagonal systems corresponding to certain rows or columns and will solve for the surface fluxes—$\bar{\phi}^y_m$'s and $\bar{\phi}^x_m$'s—and will add its contribution (for a particular direction) to each $\bar{\bar{\phi}}_m$ using the continuity Eq. (2). At the end of this calculation, each processor sends to node 0, the component of $\bar{\bar{\phi}}_m$ it has computed. (Note that node 0 also performs Step 2.)

Step 3: Node 0 receives the $\bar{\bar{\phi}}_m$'s from all processors and sums their contributions for each cell. It then takes the old $\bar{\bar{\phi}}_m$'s and compares them with the new $\bar{\bar{\phi}}_m$'s by testing the pointwise relative difference against ϵ. If the ϵ criterion is satisfied, Node 0 sends the $\bar{\bar{\phi}}_m$'s to the host which successfully terminates execution. Otherwise Steps 2 and 3 are repeated.

5. HYPERCUBE RESULTS

Three variants of the parallel algorithm described above were implemented and tested on the hypercube. In the program the new iterates for the $\bar{\bar{\phi}}_m$'s are stored in the array NEWPHI(I,J) and the old iterates in the array PHIBAR(I,J). The three programming cases are as follows:

Method A: Node 0 sends the fully dimensioned array PHIBAR(I,J) to all the nodes except itself. The nodes send the fully dimensioned array NEWPHI(I,J) to Node 0. The maximum dimension in each case is 50 × 50. Node 0 does not send NEWPHI(I,J) to itself.

Method B: Node 0 sends the fully dimensioned array PHIBAR(I,J) to all the nodes except itself. Each node sends only that row or column of the NEWPHI(I,J), (I = 1,...., n and J= 1,..., n), where the computational mesh size is $n \times n$, to Node 0.

Method C: Node 0 sends only a row or column of PHIBAR(I,J) to the node assigned to compute the tridiagonal system corresponding to that row or column. Each node sends only that row or column of the NEWPHI(I,J) to Node 0. In each case, I= 1,..., n, and J= 1,...., n. Clearly this method minimizes the size of communicated messages, but requires larger overhead operations in order to manipulate the arrays into the forms described above.

Methods A, B, and C were executed on the Intel iPSC/2 hypercube with 64 processors (Dunigan and Romine, 1987). Each processor has 4 megabytes of memory.

The measured node CPU times for different cube dimensions, i.e., number of processors, are presented for a test problem with 32 × 32 mesh in Table II. For this mesh size the largest number of independent processes is 64, so that the dimension 6 cube exploits the parallelism in this case to its limit.

Table II. Performance of methods A, B, C for 32×32 mesh (number of iterations = 1068; $\epsilon = 1.0E-04$)

Number of processors	Method A (sec)	Method B (sec)	Method C (sec)
1	874.46	862.07	859.22
2	502.74	505.83	530.11
4	353.62	333.80	363.69
8	347.75	249.20	279.61
16	483.83	212.83	235.46
32	822.83	199.20	216.46
64	1530.56	195.12	207.02

The 32×32 mesh is chosen because for the various choices of numbers of processors in Table II there is perfect load balance with the use of wrap mapping of the processors. For this mesh size, there are 32 rows and 32 columns which can be independently solved. Methods B and C in Table II show that 64 processors indeed give the minimum time, i.e., largest speedup, for the algorithm. Method A, on the other hand, gives the minimum time at 8 processors.

In Method A, the size of the messages sent is constant. For mesh sizes smaller than 8×8, the time it takes to solve the rows and columns equations (the 64 tridiagonal systems for 1068 iterations) is less than the time it takes for Node 0 to communicate to all the nodes. For 8×8 and larger mesh sizes, the communication time eventually exceeds the matrix solving portion as the number of participating processors increases for two reasons—the communication time increases as the number of processors increases, while the tridiagonal matrix solution time decreases. This communication overhead grows considerably especially for the 64 processors in Method A. Because storage of the $\bar{\bar{\phi}}_m$'s is in the form NEWPHI(I,J)

and PHIBAR(I,J), Node 0 remembers the row or column it has solved, and therefore does not send messages to itself.

Methods B and C provide better performance than A. In Method B, Node 0 sends the full array PHIBAR(I,J) to all the nodes, excluding itself. However, each node sends only that row or column of the NEWPHI(I,J) array which it has computed. Compared to Method A then, Method B involves a drastic reduction in the total size of communicated messages, hence the communication penalty is decreased markedly, resulting in monotonically increasing speedup as shown in Table II.

Method C is an extension of Method B, except for the sending of the PHIBAR(I,J) array. This time, Node 0 sends that row or column of $\bar{\bar{\phi}}_m$'s which is needed by the node performing the row or column tridiagonal system. Although Method C has reduced message sizes compared to B, the latter outperforms the former. The difference lies in the fact that in Method B there is extra overhead needed to extract the parts of the PHIBAR array that need to be communicated to each processor. Node 0 is thus sending more messages of smaller sizes. It must be noted that this overhead is very penalizing because it is performed sequentially on node 0. The overhead needed by node 0 to position the received NEWPHI contributions in their proper locations is minor in comparison, leading to the higher speedup for Method B even though it involves a larger total size of communicated messages than Method C.

Table III lists the speedup and efficiency performance of the hypercube for Method B. The speedup is defined by $S \equiv T_1/T_P$, where $T_1(T_P)$ is the CPU time required to achieve convergence by one (P) processors. The efficiency is $E \equiv (S \times 100)/P$.

Table III. Speedup and efficiency vs number of processors for the test problem with 32 × 32 mesh (Method B)

Number of processors	Speedup	Efficiency
2	1.70	85%
4	2.58	65%
8	3.46	43%
16	4.05	25%
32	4.33	14%
64	4.42	7%

6. ON THE ADEQUACY OF MESSAGE PASSING MACHINES FOR SOLVING THE NODAL DIFFUSION EQUATIONS

The standard Amdahl law (Amdahl, 1967) does not take into account the overhead due to internode communications, which is a significant factor in message passing computers. In this section we generalize Amdahl's formula to cover this case as well. We use the resulting formula to investigate the impact of communication on the performance of the parallel algorithm, and to speculate on the problem size that can take full advantage of intensively parallel machines' capabilities. The communication time depends very strongly on the parallel architecture (i.e., the node connection scheme), the solution algorithm (e.g., the number and size of communicated messages), the number of nodes participating in the calculation, as well as on the specifications of the hardware itself. We assume that the dependence on the number of participating processors, K(P), is separable from the remaining dependencies, is strictly a non-decreasing positive function of P, and that for a given machine with a specific algorithm implemented to solve a given problem, all other dependencies constitute a multiplicative constant, denoted T_c. That is, the total communication time is given by $T_c K(P)$. Let T_s, and T_p be the CPU time required to perform the serial portion and the

parallel portion of the algorithm, respectively, on one processor. Then the CPU time required to solve this problem on P processors, is given by

$$T(P) = T_s + T_p/P + T_c K(P) \; . \tag{12}$$

and the CPU time required to solve this problem on one processor, or an equivalent sequential machine, is given by:

$$T(1) = T_s + T_p \; . \tag{13}$$

It follows immediately that the speedup, $S(P)$, is given by,

$$S(P) \equiv \frac{T(1)}{T(P)} = \frac{1}{s + \frac{1-s}{P} + cK(P)} \; , \tag{14}$$

where $s \equiv T_s/(T_s + T_p)$, and $c \equiv T_c/(T_s + T_p)$. Clearly $0 \leq s \leq 1$, and $c \geq 0$. However, cases of interest must have a large parallelizable portion, so that s is close to unity, at a low communication cost, so that c is close to zero. Indeed when $c = 0$, i.e., neglect the communication penalty, we recover precisely Amdahl's law. Also the efficiency, $\eta(P)$, follows:

$$\eta(P) = \frac{1}{1 + s(P-1) + cPK(P)} \; . \tag{15}$$

The main special feature that distinguishes message-passing parallel processing computers is the possibility of using a very large number of processors. Since shared memory machines can support, in the order of 30 processors at a much lower communication cost, one would prefer a message passing architecture only in cases where large speedups can be achieved by using a large number (maybe 50 or more) of processors.

The competition, hence compromise, between message-passing and shared-memory parallel computer architectures can be seen clearly by considering the dependence of the speedup and efficiency on the number of participating processors in the ideal case of vanishing communication penalty. As well known the speedup increases asymptotically to the constant value $1/s$ as $P \to \infty$, while $\eta(P) \to 0$. The fact that the speedup increases indefinitely, albeit very slowly and costly beyond a certain point, means that there is a continuous gain to be made by using more processors. Applications may be envisioned (e.g., real time and faster than real time calculations) where increasing the speed of performing

a calculation supercedes all cost (i.e., efficiency) considerations thus giving rise to the plausability of intensively parallel, distributed memory computers. Implicit in this argument, and in fact in the derivation of all above equations is the assumption that the implemented algorithm offers a very large number of independent processes that can be executed simultaneously on P nodes for all values of P considered. Therefore in practice, even though the Amdahl model does not provide an upper limit on the number of processors used, the specific application does.

So in a very broad sense shared memory architectures are more suited for coarse grained parallel algorithms that offer only a limited number of independent processes, and also for algorithms that require extensive data sharing between the independent processes. Hence, we judge the adequacy of message passing machines to solving the algorithm described in Section 5 by examining the severity of the communication penalty and estimating the problem size that produces enough independent processes to justify using such machines.

For the case $K(P) =$ constant, the speedup again increases monotonically to saturation at $(s + cK)^{-1} < 1/s$, and the efficiency vanishes asymptotically as $P \to \infty$. When $K(P)$ is not constant, on the other hand, the speedup can attain a maximum, i.e., optimum value at $\tilde{P}$ satisfying

$$\tilde{P}^2 \frac{dK}{dP} = \frac{1-s}{c} \quad . \tag{16}$$

If $\tilde{P}$ is smaller than the number of independent processes for a given size problem, it sets a stricter limitation on the usage of message passing computers, and if in this case $\tilde{P}$ is of order 30 or smaller, it makes more sense to use a shared memory machine. That is to say the minimum requirement for a parallel algorithm to be adequately implemented on a message passing machine is that the smaller of $\tilde{P}$ and the number of independent processes be significantly larger than about 30. Additional requirements would be to achieve significant speedup at reasonable efficiency ($\sim$ 80%) with more than 30 participating processors. In the remainder of this section we determine $K(P)$, and estimte s and c for two of the three variants of the iterative algorithm described in Section 5, then we determine the adequacy of the Intel iPSC/2 hypercube for each method. The estimates are for a simple test problem on a 32×32 mesh using the accelerated iterative algorithms A and C. Analysis of method B performance (apparently the

best of the three variants) is complicated by the fact that the parallel portion of the code does not vary as $1/P$ as stipulated by Eq. (12). Moreover the behavior of $K(P)$ is erratic and can not be fit by simple functions. Hence, we exclude analysis of this method's performance.

6.1 ANALYSIS OF THE PERFORMANCE OF METHOD A

A linear function of the form

$$K(P) = P,\ P \leq 2,\ P(1) = 0 \quad , \tag{17}$$

fits the measured communication times for Method A very well, especially for large P, the range of interest here. Equations (14) and (15) immediately reduce to,

$$S(P) = P/[1 + (P-1) + cP^2] \quad , \tag{18}$$

and

$$\eta(P) = 1/[1 + s(P-1) + cP^2] \quad . \tag{19}$$

The optimal number of processors, given by Eq. (16), is

$$\tilde{P} = \sqrt{\frac{1-s}{c}} \quad , \tag{20}$$

and the corresponding speedup and efficiency are

$$\tilde{S} = [s + 2\sqrt{c(1-s)}]^{-1} \quad , \tag{21}$$

and

$$\tilde{\eta} = \frac{\sqrt{c/(1-s)}}{s + 2\sqrt{c(1-s)}} \quad , \tag{22}$$

respectively. As expected, Eq. (20) indicates that large values of $\tilde{P}$ are obtained when both s and c are very small. Indeed in the implementation of the nodal diffusion equations, we found that normally s is very small. A best case estimate therefore will result if we set $s = 0$ to see that the optimum speedup cannot exceed $1/2\sqrt{c}$ resulting in an efficiency less than 50%.

According to the criteria set forth above, we conclude that the hypercube is not adequate for algorithms of the type represented by Method A, namely $K(P) \sim P$.

6.2 ANALYSIS OF THE PERFORMANCE OF METHOD C

The measured communiction time fits very well the function,

$$\begin{aligned} K(P) &= 1 - 1/P \ , \\ T_c &= 139.96(\text{sec}) \ . \end{aligned} \tag{23}$$

Clearly this function grows much slower with P than the linear function of Method A, leading to better performance as evidenced by the results of Table II. Moreover, for this $K(P)$ the optimal number of processors predicted by Eq. (16) is unbounded, resulting in monotonically increasing speedup as in the ideal (Amdahl) case. The maximum, i.e., asymptotic, speedup in this case is,

$$\tilde{S} = 1/(s + c) \ . \tag{24}$$

For this case we found that $s = .0622$ and $c = .163$, so that $\tilde{S} = 4.44$, a relatively low speedup compared to the total number of independent processes (64). Indeed the results in Table II show the extremely low efficiencies achieved with more than two processors. Hence, it appears that Method C also does not perform adequately on the hypercube for the problem size considered here. In order to see the effect of the problem size, m, on this conclusion, we fit simple functions of m to s and c to find that $s(m) \sim .06$ and $c(m) \sim 5.78/m + .07$. Hence, the speedup, Eq. (14), becomes,

$$S(P, m) = \frac{P}{1 + (P - 1)(.13 + 5.78/m)} \ . \tag{25}$$

S increases monotonically with m and saturates as $m \to \infty$ at $P/[1 + .13(P - 1)]$, which for all P is smaller than 7.7. That is Method C speedup on the Intel iPSC/2 is bounded from above by 7.7 for all problem sizes and all numbers of participating processors. This is sufficient to deem the hypercube inadequate for solving the present algorithms for the nodal diffusion method equations. Development of other algorithms or advances in the hardware performance will be necessary before this conclusion can be revised. Meanwhile we emphasize

the adequacy of shared memory machines to the parallel solution of our new algorithm (Kirk and Azmy, 1989).

7. CONCLUSIONS

We have derived a new iterative algorithm for solving the neutron diffusion equation in two dimensional geometry that is highly parallelizable and therefore is applicable to parallel computers. We implemented our new algorithm on the Intel iPSC/2 hypercube and investigated its performance on a simple test problem. By reducing the message passing on the hypercube, total CPU time was significantly reduced. Speedups were achieved, but were found to be very small, hence, inefficient when many procesors were used. We analyzed the performance of the parallel code and extrapolated our results to larger size problems to show that Intel iPSC/2 hypercube is not adequate for solving the nodal diffusion method equations even for very large problem sizes. Shared memory machines, such as the Sequent, appear to be more suitable for this problem.

REFERENCES

1. G. Amdahl, "Validity of the Single-Processor Approach to Achieving Large-Scale Computer Capabilities," *AFIPS Conf. Proc.*, **30**, 483 (1967).

2. W. F. Ames, *Nonlinear Partial Differential Equations in Engineering*, Academic Press, New York, 1965.

3. Tony F. Chan, Youcef Saad, and Martin H. Schultz, "Solving Elliptic Partial Differential Equation on Hypercubes," *Hypercube Multiprocessors*, pp 196–210, 1986.

4. Jack Dongarra, *LINPACK Subroutines*, Argonne National Laboratory, Argonne, Ill., 1978.

5. Tom H. Dunigan and C. H. Romine, "Notes on a Short Course on the Practical Use of Multiprocessors," Engineering Physics and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, Tenn., Sept. 29–Oct. 3, 1987.

6. M. Haghoo and W. Proskurowski, "Parallel Implementation of Domain Decomposition Techniques on Intel's Hypercube," in the *Third Conf. on Hypercube Concurrent Computers and Applications, Vol. II-Applications*, Pasadena, Calif., 1988.

7. B. L. Kirk and Y. Y. Azmy, "A Parallel Approach to the Nodal Method Solution of the Two-Dimensional Neutron Diffusion Equation," presented at ANS Topical Meeting on Advances in Nuclear Engineering Computation and Radiation Shielding, Santa Fe, New Mexico, April 9–13, 1989.

8. R. D. Lawrence, *Prog. Nucl. Energy*, **17**, 271–301, 1986.

9. H. L. Rajic and A. M. Ougouag, "A Vectorized-Concurrent Nodal Neutron Diffusion Method," p 163 in *Proc. 1988 International Reactor Physics Conference, Jackson Hole, Wyoming, Sept. 18–22, 1988* Vol. IV, American Nuclear Society, LaGrange Park, Illinois, 1988.

10. Sung-Kyun Zee and Paul J. Turinsky, "Vectorized and Multitasked Solutions of the Two-Group Neutron Diffusion Equations," p 83 in *Proc. ANS Topical Meeting on Advances in Reactor Physics, Mathematics, and Computation, Paris*, Vol. III, American Nuclear Society, LaGrange Park, Illinois, 1987.

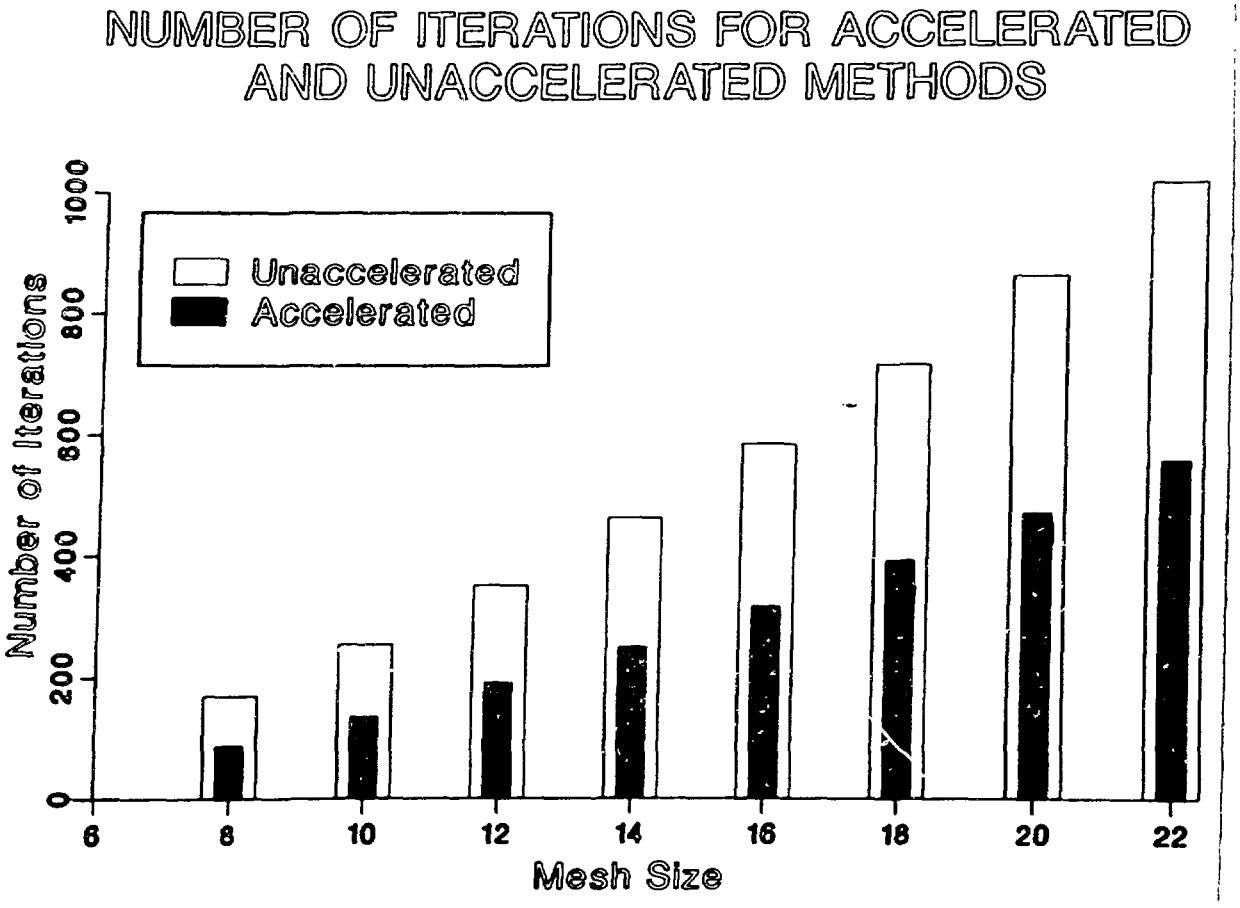
NUMBER OF ITERATIONS FOR ACCELERATED AND UNACCELERATED METHODS
Unaccelerated
Accelerated
Number of Iterations
0
200
400
600
800
1000
6
8
10
12
14
16
18
20
22
Mesh Size

Figure 1