CHAPTER 23

Parallel Efficiency of a Domain Decomposition Method

M. Haghoo*
Włodzimierz Proskurowski*

Abstract. Domain decomposition techniques for elliptic PDEs in a rectangular region are considered. By dividing the region into square boxes, the original problems are replaced by a set of subproblems. These subproblems are either independent or can be easily decoupled. More than 80% of execution time is spent to solve these independent and decoupled subsystems. Therefore this technique is well suited for parallel processing.
We study the speed-up and efficiency factors as functions of mesh size and the number of boxes. We also study the dependence of the overall execution time on the number of boxes and processors.

1. Introduction. While many algorithms have to be radically modified for parallel implementation, or the parallelism has to be realized on very low level, as in dot products of two vectors, the domain decomposition techniques are well suited for parallel processing (see also [3]).
Substructuring of the domain transforms the large original system into a set of subproblems of smaller and equal dimensions. Application of a proper preconditioner that decouples these connected subproblems increases substantially the rate of convergence. Then these subsystems lend themselves for a balanced, high level parallel processing.
We consider elliptic PDEs with Dirichlet boundary conditions in a region that is divided into subregions, subsequently called boxes. In the Neumann-Dirichlet preconditioner we partition the region in a chessboard like manner in which white and black squares correspond to Neumann and Dirichlet boxes. Half of the boxes (the Dirichlet ones) form a set of disjoint subproblems and remaining (Neumann) boxes can be easily decoupled.

*Department of Mathematics, University of Southern California, Los Angeles, CA 90089-1113

The dominant portion of the execution time is spent on solving subproblems on these boxes. Therefore, these expensive computations can be reduced substantially in a parallel implementation.
We study how well the problem is suited for parallel processing. We use Intel's hypercube of 16 nodes and large memory. We perform extensive experiments to find such parameters (number of mesh points, boxes, and processors) for which the maximum speed-up and efficiency and the minimum execution time can be achieved.

2. Problem statement. We seek the numerical solution $u(x,y)$ on $\Omega = (0,1)x(0,1)$ so that:

$$-\Delta u + cu = F \quad \text{on } \Omega,$$
$$u = G \quad \text{on } \partial\Omega.$$

By using the finite difference method and applying the 5-point stencil approximation to the Laplacian, we obtain the linear system

$$Au=f.$$

The goal is to solve this system effectively. Three factors are considered:
1. Substructuring the domain.
2. Using an effective preconditioner in the Conjugate Gradient method.
3. Implementing the problem on a parallel processor.

Figure 1 shows schematically the relation between these factors.

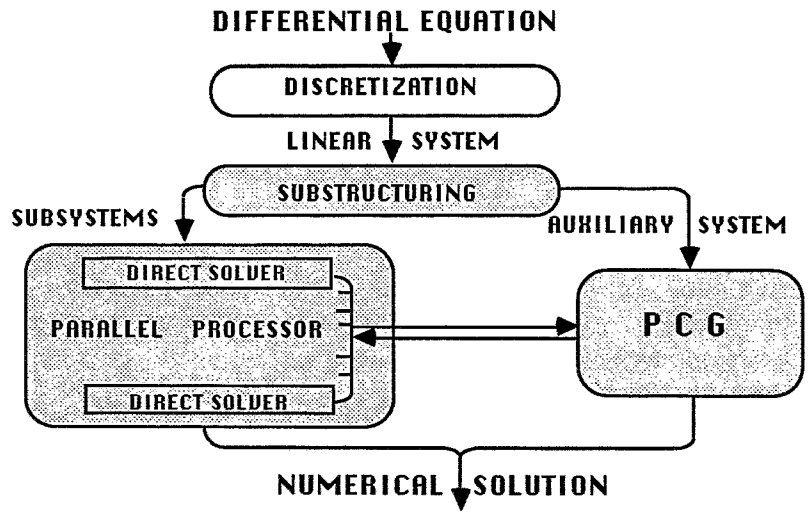

Figure 1

3. Substructuring. The domain Ω in which the problem is defined, is partitioned by mesh lines parallel to coordinate system into boxes. We consider four types of unknowns (see Figure 2):
1. Points inside white boxes,
2. Points inside shaded boxes,
3. Points on separator lines,
4. Cross points.

We call *Dirichlet boxes* all points in *1*, while we call *interconnected Neumann boxes* the totality of points belonging to *2*, *3*, and *4*. We further divide the second group into the the cross points and decopuled Neumann boxes which we call just *Neumann boxes*. This notation is used for the Neumann-Dirichlet preconditioner.

4. Preconditioned Conjugate Gradient Method. The Neumann-Dirichlet preconditioner, represented by matrix B, is a related problem to A in which the Neumann conditions are imposed on the sides of every other box, and Dirichlet conditions on the remaining boxes. Here, as shown in Figure 2, the region is partitioned so that the white squares correspond to the Dirichlet boxes, and the shaded squares together with the separator lines correspond to the Neumann boxes. The sides of Neumann boxes that coincide with the boundary of the region keep their original Dirichlet conditions.

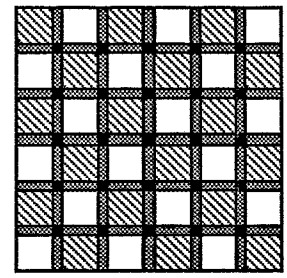

Figure 2

The original system $Au=f$ can be replaced by the following three steps:

$$Bv = h, \qquad (1)$$
$$Cw = g, \qquad (2)$$
$$Bu = h+Sw, \qquad (3)$$

where $g = S^T(f-AB^{-1}h) = S^T(f-Av)$, $C = S^TAB^{-1}S$, and $S = [0, I]^T$ is a n^2 by q matrix, n is the number of mesh points on each side of the region, and q is the number of total points in the inter-connected Neumann boxes. Here h is the same as f except on separator lines where components of h can be chosen arbitrarily.

Steps (1) and (3) are solved directly, and the capacitance system (2) is solved by the preconditioned conjugate gradient method in which B is used as a preconditioner. Forming C is extremely expensive. Fortunately, C need not to be generated. Instead, for a given vector w_j, one needs to form Cw_j in each step of the conjugate gradient iterations. Considering the fact that $Cw_j = (S^TA)B^{-1}Sw_j$, one obtains Cw_j by solving the system $Bz = Sw_j$ and then by applying S^TA to z. Thus solving (1)-(3) involves only systems with B.

It should be noted that the conjugate gradient method iterations are carried out on vectors defined only on separator points (the residuals and other vectors are zero in the interior of the boxes). Therefore, the corresponding computations are reduced to points belonging to separator lines only.

5. Sequential Implementation. As we discussed the main task is to solve a system with B repeatedly. In this section we detail solving a B-system sequentially and in section 6 we describe how to solve this system in parallel. Let us denote the matrices A and B by a 4x4 block matrices as following:

$$A = \begin{bmatrix} A_{11} & 0 & A_{13} & 0 \\ 0 & A_{22} & A_{23} & 0 \\ A_{13}^T & A_{23}^T & A_{33} & A_{34} \\ 0 & 0 & A_{34}^T & A_{44} \end{bmatrix}$$

-- Dirichlet pts
-- Neumann inner pts
-- Separator pts
-- Cross pts

$$B = \begin{bmatrix} A_{11} & 0 & A_{13} & 0 \\ 0 & A_{22} & A_{23} & 0 \\ 0 & 2A_{23}^T & A_{33} & A_{34} \\ 0 & 0 & A_{34}^T & A_{44} \end{bmatrix}$$

Note that B is different from A only in the darkened boxes.
Let A_N denote the right bottom submatrix of B, and define B_N as preconditioner for A_N :

$$A_N = \begin{bmatrix} A_{22} & A_{23} & 0 \\ 2A_{23}^T & A_{33} & A_{34} \\ 0 & A_{34}^T & A_{44} \end{bmatrix} \qquad B_N = \begin{bmatrix} A_{22} & A_{23} & 0 \\ 2A_{23}^T & A_{33} & A_{34} \\ 0 & 0 & I \end{bmatrix}$$

Then computing v in $Bv=h$ consists of solving:

$$A_N v_N = h_N \qquad (4)$$

$$A_{11} v_1 = h_1 - A_{13} v_3 \qquad (5)$$

where v_N and h_N correspond to the Neumann boxes and the cross points.
While (5) is a system of disjoint subsystems, (4) is a coupled large system and needs to be decoupled first. We, therefore, use the capacitance system once more, where the capacitance matrix C_N is defined on cross points only.
By using this preconditioner B_N , we replace (4) by three steps similar to those described in Section 4.

$$B_N z_N = k_N \qquad (6)$$

$$C_N w_N = g_N \qquad (7)$$

$$B_N v_N = k_N + S_N w \qquad (8)$$

where g_N , C_N , and S_N are defined similarly to g, C, and S in Section 4.
It is important to notice that in equations (6)-(8) the subsystems on Neumann boxes are decoupled into a set of subproblems on individual Neumann boxes, and thus all of these equations can be solved directly.
By solving (6), (7), and (8) one obtains the solution to (4). Using this solution, one solves (5), which completes the process of solving a system with B .
We now briefly describe how to solve a subproblem on a Neumann box. The Neumann boxes are of three types (Figure 3): boxes with four corners removed, boxes with two corners removed, boxes with one corner removed. Figure 4 shows the corresponding matrices of the subproblems of inner dimensions $m=3$. These matrices are banded positive definite, where half of

the bandwidth is equal to the width of corresponding box. Shifts of the off diagonals of these matrices are caused by the missing corners.

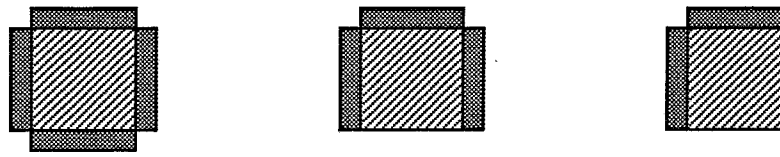

Figure 3

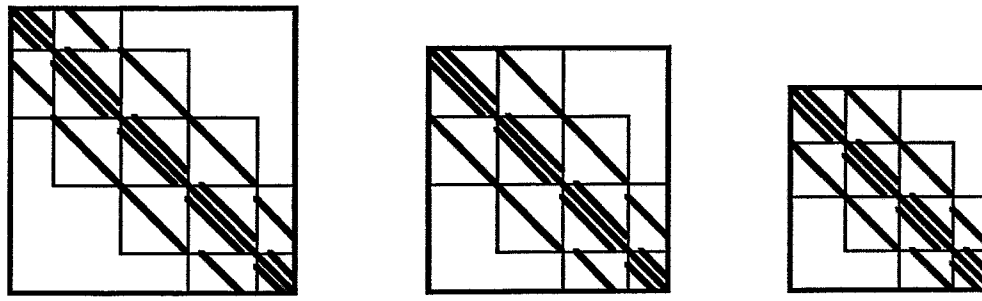

Figure 4

To solve subproblems on Neumann boxes, (as well as those on Dirichlet boxes) one can use a direct solver or a modified fast solver.
With a general purpose package like LINPACK, one can solve subproblems on individual boxes directly without further modifications. On the other hand, FISHPACK routines can be applied only to separable problems on a rectangular domain. Although our model problem **(9)** is separable, but the domain (in case of a Neumann box) is not a rectangle. One needs to modify the problem by extending the domain to a complete rectangle, and using the capacitance system at the third level. For more detail see **[1]**. Asymptotically, the computational complexity of FISHPACK routines is much smaller than those of LINPACK.

6. Parallel Implementation.

The problem is well suited for parallel processing. From previous section we can conclude that the main computation task is to solve a linear system with B, repeatedly. As discussed previously, this system consists of a set of disjoint Dirichlet subsystems and a set of Neumann subsystems inter-connected through cross points. While the first subproblems are immediately applicable for parallel processing, the latter ones can easily be decoupled. Thus, all the subproblems readily lend themselves for a balanced high level parallel processing.
The parallel algorithm is based on the sequential one. The following computations occur repeatedly in the algorithm:
(a) Solving a system with B.
(b) Small steps such as multiplying $S^T A$ by a vector, computing inner products.
We denote the number of mesh points in each coordinate direction by n, the number of boxes by n_0, the total number of separator points by s, and the size

of a box by m. Thus, the total number of points is n^2, the number of boxes is n_0^2, and the number of points in a box is m^2. This notation is used throughout the paper.

Each preconditioned conjugate gradient iteration contains one instance of operations (a) and a few instances of (b). Denote the number of operations for (b) by s. The number of separator points is bounded from above by $n_0^2 \times 2m$, since each box contributes at most $2m$ points. Thus:

$$s < n_0^2 \times 2m = (n/m)^2 \times 2m = 2n^2/m,$$

and the operation counts for (b) is proportional to n^2/m.

In contrast to (b) the cost of solving a system with B is significant and it dominates all the others combined. The operation count for a banded symmetric positive definite subsystem on a box is of order m^4 for the decomposition, and of order m^3 for the substitution. Since our model problem (9) has constant coefficients, only three different types of boxes arise, and thus three different subsystems needed to be formed, factored, and saved at the preprocessing stage with negligible cost and small amount of storage. Then, using the factored form, one needs only substitution to solve a subsystem on a box. Thus, this operation count during the iterations is reduced to m^3. Therefore, the number of operations for solving a B system is proportional to:

$$n_0^2 \times m^3 = (n/m)^2 \times m^3 = n^2 m.$$

Consequently, the ratio of the operation count for a B-solver over the other computations in each iteration is proportional to $n^2 m/(n^2/m) = m^2$, which is substantial for all values of m ($4 <= m <= 17$) used.

The experiments confirm that B-solvers are the dominant parts of the computations and show that well over 80% of CPU time is spent for B-systems (see further in Table 3). We, therefore, solve these systems in parallel.

A system with B is solved by performing (6), (7), (8), and (5), in that order, by direct methods. In this process there are three instances of data exchanges, those corresponding to (6), (8), and (5).

The parallel algorithm proceeds as follows. First, after solving the Neumann subsystems of (6), processors have to communicate to form g_N, the right hand side of (7), that is solved directly and sequentially in the host (C_N has already been formed in preprocessing steps). Then, the processors solve (8) in parallel and distribute the data for Dirichlet subsystems. Finally, after solving (5) in parallel, the processors exchange data to form the inner products for conjugate gradient iterations.

We assign one or more of the subsystems to each node of the parallel processor. The communication for solving one subsystem consists of sending the right hand side of the subsystem to a node and receiving the solution from the node.

There are some sequential computation in the preprocessing stage and in conjugate gradient iterations. The volume of these computations is relatively small.

We, therefore, can divide the computations into three categories: computations for solving systems with B, other repeated computations, such as computing inner products, adding two vectors, etc, and one-time occurring computations, such as forming and factoring C_N.

We apply parallelism only to the first category. The volume of computations of second and third ones are small. We justify this further in the next section (see also Table 3).

7. Numerical Results. The programs were developed in FORTRAN-77. The sequential version was run on a DEC-KL10, and the parallel version on a 16-node Intel's hypercube, iPSC (Intel's Personal Super Computer).
As a model problem we chose

$$-\Delta u + u = F \quad (9)$$

in (0,1)x(0,1) with Dirichlet boundary condition. The right hand side F is chosen so that the exact solution is $u = 20\sin(\pi y)x(1-x)$.
Let as before n and n_0 be the number of mesh points and boxes in each coordinate direction, respectively. We ran the sequential program for $n = 16, 24, 32, 48, 64, 96, 128$, and $n_0 = 2, 4, 6, 8, 12, 16$. All possible combinations were executed. Table 1 shows the number of iterations and estimate of the condition number of the capacitance matrix C for different values of n and n_0. The table contains the main results obtained using the sequential version.

CONVERGENCE RATE

Mesh Points	Number of Boxes	Number of Separator Points	Number of Cross Points	Number of Iterations	Estimate of Condition #
32^2	4^2	168	9	7	7.08
	8^2	336	49	8	4.91
48^2	4^2	264	9	8	8.79
	6^2	420	25	9	7.54
	8^2	560	49	8	6.39
	12^2	792	121	7	4.92
	16^2	960	225	7	4.01
64^2	4^2	360	9	8	10.11
	8^2	784	49	9	7.68
	16^2	1440	225	7	4.94
96^2	4^2	552	9	8	12.11
	6^2	900	25	10	10.81
	8^2	1232	49	9	9.56
	12^2	1848	121	9	7.63
	16^2	2400	225	8	6.45
128^2	4^2	744	9	8	13.61
	8^2	1680	49	9	11.02
	16^2	3360	225	9	7.65

Table 1

For most of our experiments we chose $u = 20\sin(\pi y)x(1-x)$. To study the effect of smoothness of the solution on the rate of convergence, we executed the same problem as in Table 1, this time with a solution randomly generated. The

number of iterations have increased by an average of 24% (with maximum deviation of 11%) . The results are in Table 2. This increase is by no means significantly large, and therefore all the rest of the experiments were carried out for the smooth solution.

ITERATION NUMBER OF SMOOTH AND RANDOM SOLUTION

Number of Mesh Points	Number of Boxes	Iteration Number of Smooth Solution	Iteration Number of Random Solution	Increment (in %)
32^2	4^2	7	9	29
	8^2	8	9	13
48^2	4^2	8	9	13
	6^2	9	12	33
	8^2	8	10	25
	12^2	7	9	29
	16^2	7	8	14
64^2	4^2	8	9	13
	8^2	9	11	22
	16^2	7	9	29
96^2	4^2	8	10	25
	6^2	10	12	20
	8^2	9	12	33
	12^2	9	11	22
	16^2	8	10	25
128^2	4^2	8	10	25
	8^2	9	12	33
	16^2	9	11	22

Table 2

We group the computations into three categories: computation for solving systems with B, computations for other repeated steps, and computation for those steps that occur only once. We show the execution time for each category (in %) in Table 3; columns 3, 4 and 5, respectively. As Table 3 shows, up to 87% (and an average of 80%) of CPU time is spent for solving the systems with B. We can conclude that the computation for these systems play a dominant role, and the rest is relatively small. We, therefore, apply parallelism only to solve the systems with B.

Let p be the number of nodes of the hypercube that are used in a parallel execution. And let, as before, n and n_0 be the number of mesh points and boxes along each side of the rectangular domain, respectively. We let these three factors vary. The only restrictions are that n be divisible by n_0 and $n_0^2/2$ be divisible by p. The first restriction is obvious. The second restriction allows us to assign an equal number of boxes to each processor (n_0^2 is the total number of boxes, half of them Dirichlet, the other half Neumann boxes). We executed the parallel version for all combinations of the sequential version, for $p = 2, 3, 4, 6, 8, 9, 12, 16$. We measured the execution times for all

different runs and compared them against the execution time of the sequential version. The results are in Table 4. The elements in this table are the ratios of the sequential time over the parallel execution time.

PARALLELIZABILITY

Number of Mesh Points	Number of Boxes	CPU time (in %) for solving B-systems	CPU time (in %) for solving Small Steps	CPU time (in %) for solving One-time steps
32^2	4^2	80	10	10
	8^2	74	20	6
48^2	4^2	82	8	10
	6^2	82	10	8
	8^2	79	15	6
	12^2	73	20	7
	16^2	64	20	16
64^2	4^2	83	5	11
	8^2	82	5	12
	16^2	71	12	16
96^2	6^2	86	7	7
	8^2	85	5	9
	12^2	82	8	10
	16^2	79	12	9
128^2	8^2	87	6	6
	16^2	82	10	10

Table 3

From Table 4, one can observe that the speed-up factors are small in the upper right portion of the table. The speed-up improves as we move to the bottom parts of the table. The standard definition of parallel efficiency, E, is given as the ratio of the speed-up over the number of nodes, $E = sp/p$, where sp represents speed-up. Table 5 shows these efficiency factors (in %) based on the data from Table 4.

There are three factors that affect the size of the entries in Tables 4 and 5: the number of mesh points, the number of boxes, and the number of processors, denoted by n, n_0, and p, respectively. Below we discuss the effects n and n_0 on speed-up and efficiency. We study the influence of each one while the other two variables are kept fixed. The dependence of speed-up and efficiency as functions of p is discussed in [2] (see also [4]).

First, we study the efficiency as a function of n while keeping n_0 and p constant. We find that the efficiency is an increasing function on n. This can be explained as following. In the complexity count the highest order term, m^3, is connected with the cost of B-solvers. Other parts of the algorithm (an overhead) are of lower order. Thus, for fixed n_0, as n (and consequently m) increases, the share of the B-solver is increased. This is confirmed by the experimental data in Table 3. The portions of the algorithm connected with B-solvers are implemented in parallel, hence higher speed-up and efficiency are achieved as n grows. Moreover, let us define r, as the ratio of

communication-time over execution-time for the portion of the algorithm which is implemented in parallel. We analyze the influence of n on r. The cost of solving a subsystem on a box (using SPBSL of LINPACK) is proportional to m^3, and the communication exchange is proportional to the size of the right hand side, namely m^2. Therefore, r is proportional to $1/m = n_0/n$ (as $m = n/n_0$). For a fixed n_0, as n increases, r decreases and, therefore, the speed-up and efficiency improve. Figure 5 shows the speed-up as a function of n for fixed n_0 and p.

SPEED-UP FACTORS

Number of Mesh Points	Number of Boxes	Number of processors: 2	4	8	16
32^2	4^2	1.48	2.07	2.07	—
	8^2	1.35	1.72	1.72	1.29
48^2	4^2	1.59	2.22	2.68	—
	8^2	1.51	2.11	2.58	2.16
	16^2	1.29	1.58	1.78	1.61
64^2	4^2	1.65	2.36	3.00	—
	8^2	1.59	2.32	2.93	3.04
	16^2	1.38	1.82	2.14	2.14
96^2	8^2	1.66	2.38	3.08	3.90
	16^2	1.52	2.14	2.71	2.95
128^2	8^2	—	2.56	3.54	4.45
	16^2	—	2.35	3.10	3.57

TABLE 4

Next, we fix n and p and discuss the affect of n_0 on efficiency. Here we see that as n_0 increases, the efficiency becomes smaller. Again, examining Table 3, we see that as n_0 increases, the percentage of CPU time required to solve B systems decreases, therefore the time percentage of parallel parts becomes smaller and thus efficiency decreases. Additionally, r, the communication overhead, is proportional to n_0/n. Thus, r increases together with n_0 and, as a consequence, efficiency deteriorates.

Thus, we conclude that speed-up and efficiency are increasing functions of n and decreasing functions of n_0. Here, a few comments are in place. One may get a wrong impression that having a large number of boxes is a disadvantage. This is not the case. In fact, from the point of view of the overall CPU time the opposite is true. The total execution time decreases as the number of boxes increases. This is true for all sizes of our problem in the sequential version and also in the parallel version (except when n is small, and n_0 and p are simultaneously approaching their extreme values). In all, increasing n_0 reduces the execution time and this reduction is substantial for large n and small p. Moreover, a large n_0, and consequently small m, reduces the storage requirements needed to solve subsystems on the boxes. And finally, for large values of p, one must have at least as many boxes as processors. Figure 6 shows speed-up as a function of n_0 for fixed n and p.

PARALLEL EFFICIENCY, in %					
Number of		Number of processors			
Mesh Points	Boxes	2	4	8	16
32^2	4^2	74	52	26	—
	8^2	68	43	22	8
48^2	4^2	80	56	34	—
	8^2	76	53	32	14
	16^2	65	40	22	10
64^2	4^2	83	59	38	—
	8^2	80	58	37	19
	16^2	69	46	27	13
96^2	8^2	83	60	39	24
	16^2	76	54	34	18
128^2	8^2	—	64	44	28
	16^2	—	59	39	22

Table 5

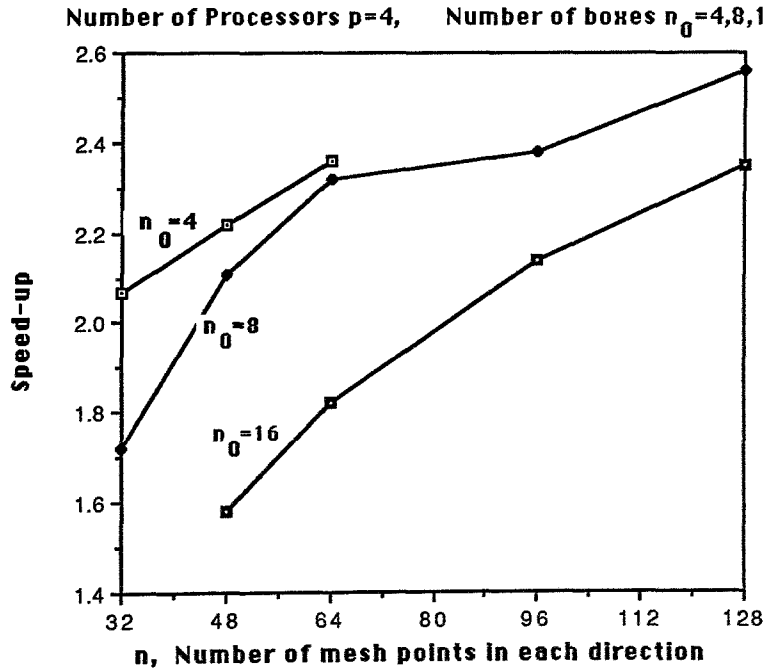

Figure 5

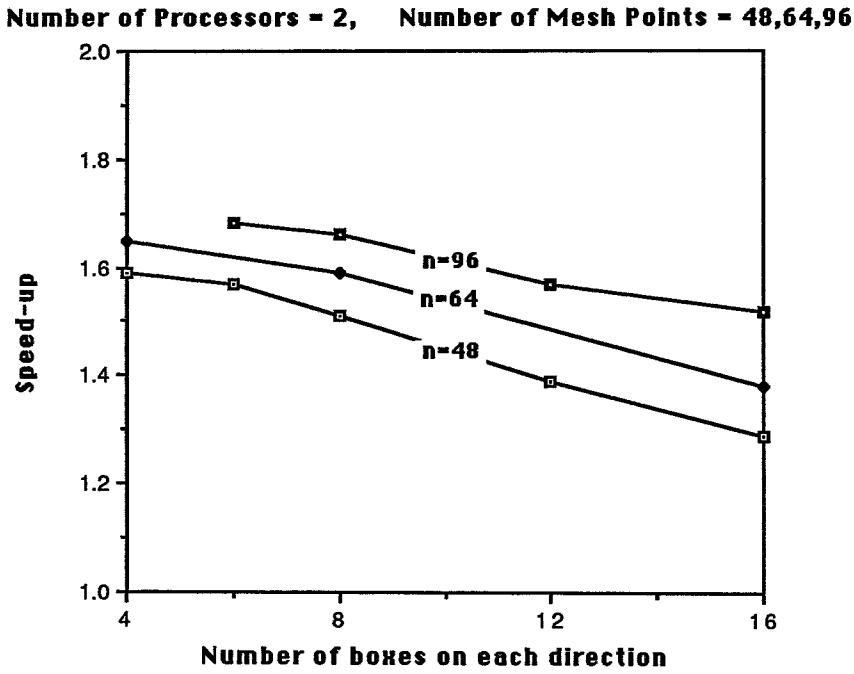

Figure 6

Efficiency substantially decreases with p. The decrease is even sharper for small problems (see [2]).

To summarize our discussion: The efficiency improves as n increases, and deteriorates as n_0 and p become larger. On the other hand, the total execution time decreases as these three factors increase, except when n_0 and p approach simultaneously their extreme values while n is relatively small. We should note that the given problem defines the value of n, and one needs only to choose the values of n_0 and p that minimize the execution time.

7. Conclusions. The results obtained using the sequential version (Table 1) are consistent with the similar results in [1].
The main conclusions obtained are:
1. The problem is well suited for parallel processing (Table 3). The speed-up factor for two processors is as high as 1.65, and the speed-up is larger than $\sqrt{p}$, in most cases, where p is the number of processors (Table 4) .
2. Efficiency is an increasing function on n and a decreasing function of n_0 and p (Table 5).
3. For a given problem of size n, the total execution time reduces as n_0 and p increase, unless n is relatively small and n_0 together with p have their extreme values.

References

[1] M. Dryja, W. Proskurowski, and O. Widlund, *Numerical experiments and implementation of domain decomposition method with cross points for the model problem*, in "Advances in computer methods for PDEs-VI", IMACS, 1987, pp. 23-27.

[2] M. Haghoo and W. Proskurowski, *Parallel implementation of domain decomposition techniques on Intel's hypercube*, Proceedings of the third conference on hypercube, ACM Publications, 1988.

[3] D. E. Keyes and W. D. Gropp, *A comparison of domain decomposition techniques for elliptic partial differential equations and their parallel implementations*, SIAM J. Sci. Stat. Comput., v. 8, No. 2, March 1987, pp. s166-s202.

[4] O. A. McBryan and E. Van de Velde, *Hypercube algorithms and implementations*, SIAM J. Sci. Stat. Comput., v. 8, No. 2, March 1987, pp. s227-s286.