Abstract: Several preconditioned conjugate gradient (PCG)-based domain decomposition techniques for self-adjoint elliptic partial differential equations in two dimensions are compared against each other and against conventional PCG iterative techniques in serial and parallel contexts. We consider preconditioners that make use of fast Poisson solvers on the subdomain interiors. Several preconditioners for the interfacial equations are tested on a set of model problems involving two or four subdomains, which are prototypes of the stripwise and boxwise decompositions of a two-dimensional region. Selected methods have been implemented on the Intel Hypercube by assigning one processor to each subdomain, making use of up to 64 processors. The choice of a "best" method for a given problem depends in general upon: (a) the domain geometry, (b) the variability of the operator, and (c) machine characteristics such as the number of processors available and their interconnection scheme, the memory available per processor, and communication and computation rates. We emphasize the importance of the third category, which has not been as extensively explored as the first two in the domain decomposition literature to date.

A Comparison of Domain Decomposition Techniques for Elliptic Partial Differential Equations and their Parallel Implementation

David E. Keyes and William D. Gropp

Research Report YALEU/DCS/RR-448
December 1985

The first author was supported by Office of Naval Research Contract #N00014-82-K-0184. The second author was supported in part by Office of Naval Research Contract #N00014-82-K-0184, National Science Foundation Grant MCS-8106181, and Air Force Office of Scientific Research Contract AFOSR-84-0360.

Keywords: *Domain decomposition, substructuring, elliptic problems, Schur complement, preconditioning, parallel algorithms.*

1. Introduction

A number of methods based on domain decomposition have been proposed in recent years for the numerical solution of elliptic partial differential equations. Such methods are based upon the observation that the domain of problem definition may be regarded as the union of two or more subdomains, on each of which the restriction of the original problem may take on a particularly convenient form. Decomposition by domain also provides a natural route to parallelism. For some problems, these methods can be interesting even as serial algorithms, when the advantages that arise from isolating the subproblems can be made to outweigh the extra work involved in enforcing the proper conditions at the interfaces of the subdomains. *A fortiori*, they are interesting as parallel methods since reasonably large independent subtasks can readily be identified. We compare the performance of several domain decomposition methods and a class of undecomposed domain methods on a common set of two-dimensional, linear, scalar problems, and examine the parallelizability of each. Our aims are to point out the unity under certain conditions of methods which have been presented independently, and also to identify some problem characteristics which tend to favor certain methods over others in the context of parallelism.

Apart from the advantage of reformulating a large discrete problem as a collection of smaller problems which can be solved independently, there are at least two other motivations for considering domain decomposition methods, which may be present individually or in combination with the first in any given problem. All are of the "divide-and-conquer" type. Domains of irregular shape can be decomposed into subdomains of regular shape on which tensor-product-based discretization schemes can be employed, leading to discrete operators of regular structure. Also, regions of relative nonuniformity of the differential operator, whether due to coefficient variability or even substantially different physics, can be isolated into different subdomains, again resulting in exploitable locally regular structure. An example of each of these motivations is given.

The decomposition topologies considered involve both simple interfaces (with and without overlap regions) and cross-points. We compare Schur complement matrix methods (*e.g.*, [1, 13, 17]), full partitioned matrix methods based on block Gaussian elimination not making explicit use of the reduced Schur complement system (*e.g.*, [4]), and other methods of variational type (*e.g.*, [16]). These classes of methods are similar at the discrete level, employing preconditioned conjugate gradient (PCG) iterations as the outer loop, and an exact equivalence between the iterates of the first two can be established under certain conditions. In all of these methods the largest implicit problems are Dirichlet or Neumann solves over the subdomains; therefore an easy handle on parallelism can provided from an *a priori* dissection of the grid. However, global communication is required in forming the inner products of the PCG iterations (and also in one class of preconditioning methods), so the optimum parallel implementation is not completely straightforward. The optimum number of subdomains is generally both architecture- and problem-dependent, since the communication cost per iteration and the overall number of iterations tend to increase with the number of subdomains. We consider the standard communication topologies of a ring, a two-dimensional mesh, and an n-cube. Onto these we consider the natural decomposition mappings: a decomposition into strips onto the ring, a decomposition into boxes onto the mesh, and decompositions of both types onto the n-cube. Serial complexity comparisons and parallel efficiency comparisons of the methods are presented.

A variety of preconditioners have been proposed, for some of which there exist theoretical results showing the convergence rate of the iterations to be asymptotically independent of the spatial resolution, or only weakly dependent thereon. Some of these optimal preconditioners are exact for uniform operators, and in practice work best when the operator coefficients do not vary too much *along* the interfaces. For problems in which there is variation along the interfaces, the condition number of the system, though asymptotically independent of resolution, is larger and

it is interesting to consider alternative low-bandwidth approximations to the Schur complement matrix, at least for sufficiently low resolution. We designate this type of preconditioning Modified Schur Complement (MSC), and include some examples in our comparisons.

The outline of this paper is as follows. In §2 we review some recent contributions to the domain decomposition literature, summarizing methods and key theoretical results, and introducing MSC preconditioning. Section 3 contains an experimental comparison of various methods on the same small-scale model problems, all implemented serially. Parallel implementation issues for domain decomposition methods are discussed in §4, followed by some preliminary experimental results generated on the Intel Hypercube. A theoretical model for the efficiency of various parallel implementations of domain decomposition algorithms on some model large-scale architectures is presented in §5, and we draw some conclusions in §6.

2. A review of recent PCG-based domain decomposition methods

The substructuring of elliptic partial differential equations by domain has served as a practical computational technique for over twenty years (*e.g.*, [23]), and its theoretical origins (as a method of proving the solvability of the Dirichlet problem on irregular regions) extend back to the last century [28]. A briefly annotated bibliography of various direct and iterative approaches is contained in the introduction of [1]. Here we summarize only recent domain decomposition algorithms which make use of preconditioned conjugate gradient iteration in the outer loop, which appear to have originated with [9]. There are other types of domain decomposition-related methods such as Schwarz-Jacobi [26] and Schwarz-multigrid [21] which are not considered in this paper.

2.1. Problem definition

The methods will be illustrated on a model second-order, positive definite, self-adjoint elliptic Dirichlet problem on a bounded domain in R^2 with a piecewise smooth boundary:

$$\begin{aligned} Lu &= f \text{ in } \Omega, \\ u &= 0 \text{ on } \Gamma \equiv \partial\Omega, \end{aligned} \tag{2.1a}$$

in the weak formulation, in which we seek $u \in H_0^1(\Omega)$ such that for all $v \in H_0^1(\Omega)$

$$A_\Omega(u,v) = (f,v), \tag{2.1b}$$

where

$$A_\Omega(u,v) \equiv \sum_{p,q=1}^{2} \int_\Omega a_{pq}(x) \frac{\partial u}{\partial x_p}\frac{\partial v}{\partial x_q}\, dx$$

and

$$(f,v) \equiv \int_\Omega f v\, dx.$$

Given a triangulation of Ω, we define the discrete subspace of H_0^1 consisting of piecewise linear functions vanishing on Γ, H_{0h}^1 . Note that the dimension of H_{0h}^1 is the number of vertices in the interior of Ω, defined as n . For all of the algorithms to be described but one, it is sufficient to consider the usual cardinal basis for H_{0h}^1, consisting of C^0 piecewise linear functions of the smallest possible support, denoted $\{\psi_j\}_{j\in N}$, $N = \{1,2,\ldots,n\}$. The discrete approximation to u in H_{0h}^1 is then represented by a vector of nodal coefficients, u_h .

The Galerkin formulation of (2.1b) leads to the matrix equation

$$Au_h = f_h, \tag{2.2}$$

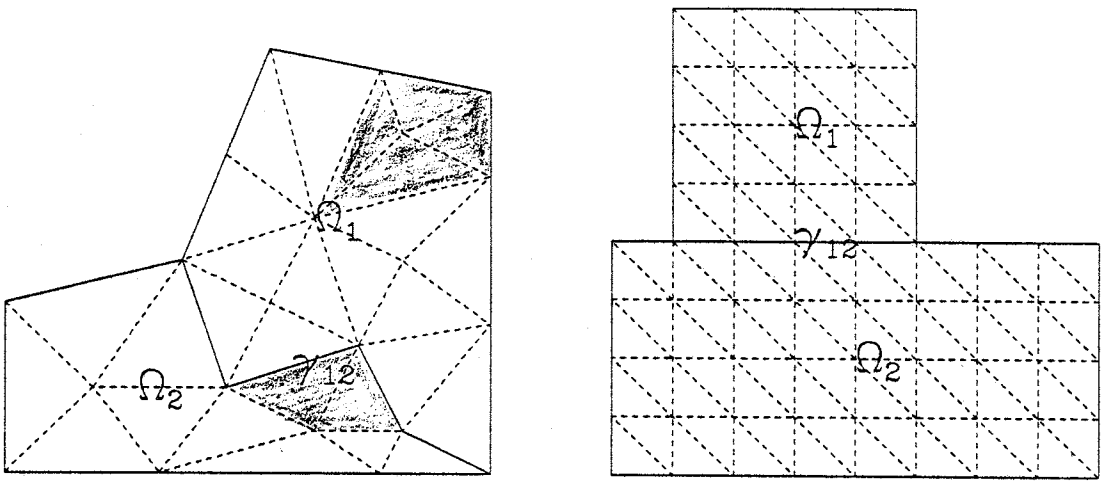


Figure 1: Sample domains illustrating the triangulation and substructuring described in §2.1. (a) A general domain showing partitioning into two subdomains with a common interface (or separator set) lying along segments of the triangulation. Support of typical nodal basis functions $\psi_{1j}, j \in N_1$ and $\psi_{2j}, j \in N_0$ are shaded. (b) A model geometry for the use of fast Poisson solvers on the subdomains: a union of uniformly triangulated rectangles.

where

$$[A]_{ij} = \sum_{p,q=1}^{2} \int_{\Omega} a_{pq} \frac{\partial \psi_j}{\partial x_p} \frac{\partial \psi_i}{\partial x_q} \, \mathrm{dx}, \quad i,j \in N$$

and

$$[f_h]_i = \int_{\Omega} f \psi_i \, \mathrm{dx}, \quad i \in N.$$

The simplest decomposition on which all of the methods can be compared is that involving two simply connected subdomains. Though somewhat academic from the point of view of parallel processing, we consider this case first because it generalizes straightforwardly to multiple-strip decompositions and allows presentation of the basic ideas without cumbersome notation. For all of the algorithms to be described but one, the intersections of the subdomains are restricted to be interfaces of lower dimension. Refering to Figure 1a, we define $\gamma_{12} = \partial\Omega_1 \cap \partial\Omega_2$.

The triangulation must be such that the segments of γ_{12} coincide with sides of the triangular elements. Let f_k denote the restriction of f to Ω_k . With a slight abuse of the usual notation, we define the discrete subspaces of functions over each subdomain which vanish on the outer boundary, but may be nonvanishing on the interface boundary, H^1_{kh}, $(k = 1, 2)$, and their subsets, H^1_{0kh}, of functions which vanish also on γ_{12} . The H^1_{0kh} have cardinal bases $\{\psi_{kj}\}_{j \in N_k}$, where N_k is an index set for the nodes interior to Ω_k, of dimension n_k . Denote by $\{\psi_{kj}\}_{j \in N_0}$, where N_0 is an index set of dimension n_0 for the nodes on the interface, the bases for the functions in H^1_{kh} which vanish at the interior nodes of Ω_k, but have nonvanishing trace on γ_{12} .

Equation (2.2) can be symmetrically permuted into the form

$$\begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \begin{pmatrix} u_0 \\ u_1 \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \end{pmatrix}, \tag{2.3}$$

where the first block row corresponds to the unknowns defined at the nodes of the separator set γ_{12}, and where A_{11} is itself a 2×2 block diagonal matrix, one block corresponding to the nodal values in each subdomain. In (2.3) and hereafter the subscript h is dropped where there is no ambiguity between the continuous and discrete formulations. The matrix A is symmetric and positive definite, as are its diagonal blocks, by the hypotheses preceding (2.1). Consequently, the interior unknowns may be formally eliminated in forming the Schur complement system (sometimes denoted the ***capacitance*** system)

$$Cu_0 = g, \tag{2.4}$$

where $C = A_{00} - A_{01}A_{11}^{-1}A_{10}$ and $g = f_0 - A_{01}A_{11}^{-1}f_1$.

As the Schur complement of A_{00} in A, C is also symmetric and positive definite. Construction of the right-hand side of (2.4) requires one solve on each subdomain with homogeneous Dirichlet conditions on the interface. Having solved (2.4) for u_0, the interior unknowns can be recovered from the lower block row of (2.3), again at the cost of one solve on each subdomain with u_0 as inhomogeneous Dirichlet data.

In the two-subdomain case it is helpful for illustrative purposes to rewrite (2.3) in the expanded form

$$\begin{bmatrix} A_{00} & A_{01}^{(1)} & A_{01}^{(2)} \\ A_{10}^{(1)} & A_{11}^{(1)} & 0 \\ A_{10}^{(2)} & 0 & A_{11}^{(2)} \end{bmatrix} \begin{pmatrix} u_0 \\ u_1^{(1)} \\ u_1^{(2)} \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1^{(1)} \\ f_1^{(2)} \end{pmatrix}, \tag{2.5}$$

where $u_1^{(1)}$ and $u_1^{(2)}$ are the unknowns in subdomains 1 and 2, respectively, and to decompose the interface diagonal block as

$$A_{00} = A_{00}^{(1)} + A_{00}^{(2)},$$

where

$$[A_{00}^{(k)}]_{ij} = \sum_{p,q=1}^{2} \int_{\Omega_k} a_{pq} \frac{\partial \psi_j}{\partial x_p} \frac{\partial \psi_i}{\partial x_q} \, \mathrm{dx}, \quad i,j \in N_0.$$

The dimensions of $A_{11}^{(1)}$ and $A_{11}^{(2)}$ are n_1 and n_2, respectively, with $n = n_0 + n_1 + n_2$.

The case of Poisson's equation on a union of rectangles, uniformly triangulated as in Figure 1b, will be frequently considered in the sequel. For this so-called Courant triangulation, fast Poisson solvers [29] may be used to invert the $A_{11}^{(k)}$.

Since it explicitly involves the inverses of the $A_{11}^{(k)}$, the matrix operator C can be expensive to construct, generally requiring n_0 solves on each subdomain. Historically, domain decomposition was first approached in this way [23], and it remains a useful procedure when the formation of the factors of C can be amortized over a large number of right-hand sides, for instance when the same discrete linear system arises at successive steps in a time-dependent problem (*e.g.*, [22]).

For cases in which the construction of C cannot be so amortized, for instance when a new system like (2.2) arises at each step in a nonlinear problem, more efficient methods have been developed by means of preconditioned conjugate gradient iteration, which at each step require a matrix-vector multiply involving C, without its explicit construction. Each PCG iteration still requires one solve on each subdomain, so the effectiveness of the PCG-based techniques depends on

keeping the number of steps small through a preconditioning which is considerably less expensive to construct and apply than C^{-1} itself.

Of course, PCG iterations can also be used to solve the full systems in the form of (2.2) and (2.3), the type of preconditioning which is natural and efficient being different in each case. In subsequent sections we make some comparisons the effectiveness and generality of techniques which take each of these three formulations as their starting points. We refer to methods which depart directly from (2.2) as global matrix methods (GMM), from (2.3) as partitioned matrix methods (PMM), and from (2.4) as Schur complement matrix methods (SCM). For subsequent reference to individual steps, we present below the form of the PCG algorithm used in applications for solving the symmetric $n \times n$ linear system $\mathcal{A}x = b$ with (symmetric) preconditioner $\mathcal{B}$:

Algorithm PCG

Choose initial iterate:

$$x^0, \text{arbitrary} \qquad (PCG.1)$$

Compute initial residual:

$$r^0 \leftarrow b - \mathcal{A}x^0 \qquad (PCG.2)$$

Compute preconditioned residual:

$$s^0 \leftarrow \mathcal{B}^{-1}r^0 \qquad (PCG.3)$$

Initialize direction:

$$p^0 \leftarrow s^0 \qquad (PCG.4)$$

Compute $\mathcal{B}$-inner-product:

$$\gamma^0 \leftarrow (r^0, s^0) \qquad (PCG.5)$$

For $k = 0$ **Step** 1 **Until** *Convergence*, **Do**

Compute matrix-vector product:

$$q^k \leftarrow \mathcal{A}p^k \qquad (PCG.6)$$

Compute $\mathcal{A}$-inner-product:

$$\tau^k \leftarrow (p^k, q^k) \qquad (PCG.7)$$

Compute step length:

$$\alpha^k \leftarrow \gamma^k/\tau^k \qquad (PCG.8)$$

Update solution:

$$x^{k+1} \leftarrow x^k + \alpha^k p^k \qquad (PCG.9)$$

Compute new residual:

$$r^{k+1} \leftarrow r^k - \alpha^k q^k \qquad (PCG.10)$$

Compute new preconditioned residual:

$$s^{k+1} \leftarrow \mathcal{B}^{-1}r^{k+1} \qquad (PCG.11)$$

Compute $\mathcal{B}$-inner-product:

$$\gamma^{k+1} \leftarrow (r^{k+1}, s^{k+1}) \qquad (PCG.12)$$

Compute orthogonalization coefficient:

$$\beta^k \leftarrow \gamma^{k+1}/\gamma^k \qquad (PCG.13)$$

Update direction:

$$p^{k+1} \leftarrow s^{k+1} + \beta^k p^k \qquad (PCG.14)$$

End For

In addition to the storage and workspace requirements for the application of $\mathcal{A}$ and $\mathcal{B}^{-1}$, the algorithm requires storage for four vectors of length n. As is well known (*e.g.*, [9]), the $\mathcal{A}$-norm of the error at the kth iteration of PCG is bounded according to

$$\frac{\| x^k - x \|_{\mathcal{A}}}{\| x^0 - x \|_{\mathcal{A}}} \leq 2 \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^k,$$

where κ is the condition number of $\mathcal{B}^{-1}\mathcal{A}$, and the algorithm produces exact convergence in a number of iterations which is at most the number of distinct eigenvalues of $\mathcal{B}^{-1}\mathcal{A}$. The operators $\mathcal{A}$ and $\mathcal{B}$ are said to be *spectrally equivalent* if there exist positive constants γ_0 and γ_1 independent of the discretization such that for all x, $\gamma_0(x, \mathcal{B}x) \leq (x, \mathcal{A}x) \leq \gamma_1(x, \mathcal{B}x)$.

2.2. Schur Complement Methods (SCM)

The Schur complement methods are realized by taking $\mathcal{A}$ in Algorithm PCG to be C and selecting appropriate $\mathcal{B}$. The iterations occur on vectors of length n_0. Note that by construction the product Cp^k consists of the residuals at the nodes along γ_{12} of the original discrete operator applied to the vector which satisfies the discrete equations with homogeneous boundary conditions on Γ in each of the subdomains and equals p^k on γ_{12} . Four related choices of $\mathcal{B}$ are reviewed in this subsection.

Dryja [13, 14] showed that C of (2.4) is spectrally equivalent to the matrix $K^{1/2}$, where K is the tridiagonal matrix of order n_0 with diagonal elements 2 and off-diagonal elements -1, the discrete Laplacian operator over a uniform grid in one dimension. For notational convenience, we define the average nodal spacing along the interface, $h = (n_0 + 1)^{-1}$. Inserting a factor of 2 for consistent scaling across the methods to follow, Dryja's preconditioner has the eigendecomposition

$$M_D \equiv 2K^{1/2} = W \Lambda_D W^T,$$

where

$$[W]_{ij} = \sqrt{2h} \sin ij\pi h,$$

and

$$\Lambda_D = diag(\lambda_j^D),$$

where

$$\lambda_j^D = 2\sqrt{\sigma_j},$$

with

$$\sigma_j = 4 \sin^2 \frac{j\pi h}{2}.$$

The condition number of $K^{1/2}$, which is equal to λ_{n_0}/λ_1, grows in proportion to the number of interfacial unknowns n_0 as this number becomes large. Through their spectral equivalence, the same is true of C, which implies that the convergence of the unpreconditioned Schur complement system

iteration will deteriorate as the mesh is refined. The action of M_D^{-1} on a vector can be computed with a pair of sine transforms in $O(n_0 \log n_0)$ operations, hence the cost of the preconditioning is inexpensive in comparison with the cost of forming a matrix multiplication with a general C, which involves two-dimensional subdomain solves.

By means of FFT's in one-dimension, Dryja also gave an $O(n_0 \log n_0)$ method for performing the subdomain solves for the case $L = -\Delta$ and Ω a union of two rectangular subregions, by exploiting the regular sparsity pattern of the right-hand sides arising during the PCG iterations. In this case the overall operation count for solving (2.2) is dominated by the pre- and post-processing involved in forming g of (2.4) and backsolving, instead of by solving the Schur complement system. For more general operators and domain geometries, the cost of one PCG iteration is comparable to the pre- and post-processing.

Golub and Mayers [17] arrived heuristically at a preconditioner for C by starting from the observation that the elements $[C]_{ij}$ may often be well approximated as constant along a diagonal. In other words, the influence of the data at interfacial node j on the residual at interfacial node i depends (approximately) only on the discrete distance along the interface, $|i-j|$. This led Golub and Mayers to solve a discrete infinite domain Laplace problem with an infinite interface dividing two half-planes, the data on the interface and at infinity prescribed to be zero everywhere except at the origin, where it was taken as one. The ijth element of their preconditioner of Toeplitz form was then defined as the residual of the discrete Laplacian at the point on the interface at a distance $|i-j|$ from the origin. This preconditioner was tested and found superior to Dryja's on the problem considered in [17]. Though computationally complicated to construct, they noticed that a second, FFT-implementable preconditioner could be derived by replacing a term in their generating function expression for the interface residual by Dryja's preconditioner. Their result,

$$M_G = W \Lambda_G W^T,$$

where

$$\Lambda_G = diag(\lambda_j^G),$$

where

$$\lambda_j^G = 2\sqrt{\sigma_j + \frac{\sigma_j^2}{4}},$$

or

$$M_G = 2(K + \frac{1}{4}K^2)^{1/2},$$

has been also employed as an effective refinement of M_D in other investigations [1, 11].

Bjorstad and Widlund [1] showed that C, $C^{(1)} \equiv A_{00}^{(1)} - A_{01}^{(1)}(A_{11}^{(1)})^{-1}A_{10}^{(1)}$, and $C^{(2)} \equiv A_{00}^{(2)} - A_{01}^{(2)}(A_{11}^{(2)})^{-1}A_{10}^{(2)}$, are all spectrally equivalent. (Note that $C = C^{(1)} + C^{(2)}$.) Assuming that Ω is decomposed in such a way that it is computationally convenient to solve Neumann problems on one of the subdomains, say Ω_1, with zero Dirichlet data on $\partial\Omega_1 \cap \Gamma$ and natural boundary conditions on γ_{12} , they proposed $C^{(1)}$ as a preconditioner for C, acting on a suggestion of Dryja's. Applying $(C^{(1)})^{-1}$ to a vector p_0 of dimension n_0 requires solving the Ω_1 Neumann problem

$$\begin{bmatrix} A_{00}^{(1)} & A_{01}^{(1)} \\ A_{10}^{(1)} & A_{11}^{(1)} \end{bmatrix} \begin{pmatrix} q_0 \\ q_1^{(1)} \end{pmatrix} = \begin{pmatrix} p_0 \\ 0 \end{pmatrix},$$

whence $q_0 = (C^{(1)})^{-1}p_0$. In the case of Poisson's equation on the union of two rectangular uniformly gridded regions, as pictured in Figure 1b, an explicit FFT-implementable expression for $C^{(1)}$ was

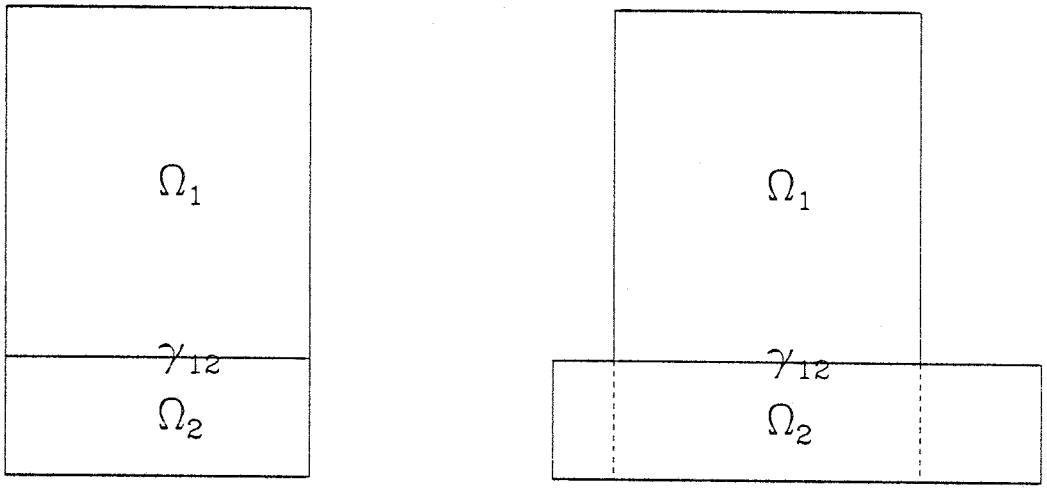


Figure 2: The applicability of the Chan preconditioner. (a) A model geometry for which $M_C = C$. (b) A model geometry for which M_C is not exact but furnishes a useful aspect ratio-sensitive preconditioner.

derived in [1]. We shall define M_B as twice $C^{(1)}$ in what follows for purposes of comparison. Let m_1 be the number of internal grid points in the vertical direction in Ω_1 . Then

$$M_B = W \Lambda_B W^T,$$

where

$$\Lambda_B = diag(\lambda_j^B),$$

where

$$\lambda_j^B = 2 \left(\frac{1 + \rho_j^{m_1+1}}{1 - \rho_j^{m_1+1}} \right) \sqrt{\sigma_j + \frac{\sigma_j^2}{4}},$$

where

$$\rho_j = r_{j-}/r_{j+},$$

and

$$r_{j\pm} = 1 + \frac{\sigma_j}{2} \pm \sqrt{\sigma_j + \frac{\sigma_j^2}{4}}.$$

This is a diagonal rescaling of M_G which takes account of the aspect ratio, m_1/n_0, of one of the subdomains. In the case where Ω is a rectangle and Ω_1 and Ω_2 are symmetrically disposed about the interface, $C^{(1)} = C^{(2)}$, $M_B = C$, and the Bjorstad-Widlund method converges in one step.

Chan has carried this circle of ideas further for the case where Ω is a rectangle and L is the Laplacian by finding the Fourier decomposition of C itself [6]. Referring to Figure 2a, let m_2 be the number of internal grid points in the vertical direction in Ω_2.

Then in the previous notation,

$$M_C = W \Lambda_C W^T,$$

where

$$\Lambda_C = diag(\lambda_j^C),$$

where

$$\lambda_j^C = \left(\frac{1+\rho_j^{m_1+1}}{1-\rho_j^{m_1+1}} + \frac{1+\rho_j^{m_2+1}}{1-\rho_j^{m_2+1}} \right) \sqrt{\sigma_j + \frac{\sigma_j^2}{4}}.$$

By taking account of the aspect ratio of both of the subdomains, Chan's method converges in one step even in the unsymmetric case, and thus may be regarded as a direct fast Poisson solver. Of course, fast Poisson solvers not making use of domain decomposition can already be applied when Ω is a rectangle, and the serial computational complexity of Chan's approach can be made nearly equal to that of these algorithms by solving the Schur complement system in Fourier transform space [24]. However, Chan's preconditioner may also be used to advantage in a more general geometry like that of Figure 2b, for which a fast Poisson solver is not available, although exact only for the inscribed $n_0 \times (m_1 + m_2)$ rectangle.

In aid of visualizing these preconditioners, surface plots of their elements as a function of their indices are given in Figure 3 below for the case $n_0 = 15$, $m_1 = m_2 = 7$.

Corresponding estimates of the condition numbers $\kappa(M^{-1}C)$, where C derives from the Laplacian on the unit square, with 16 subintervals on a side are listed along with convergence data in the second row of Table 1 in §3.

The Chan and Bjorstad-Widlund methods are both exact for the Laplacian in a symmetrically decomposed rectangular domain. For a given h, the Golub-Mayers method coincides with these two methods in the infinite aspect ratio limit of the domain geometry:

$$\lim_{\substack{m_1 \to \infty \\ m_2 \to \infty}} \lambda_j^C = 2\sqrt{\sigma_j + \frac{\sigma_j^2}{4}} = \lambda_j^G.$$

The Dryja preconditioner is not exact for any domain geometry limit. All four of the preconditioners discussed above are *optimal* for the two-subdomain case in the sense that the condition number of the Schur complement system approaches a constant independent of h as the mesh is refined with fixed geometry.

A disadvantage of working with the Schur complement system, independent of preconditioner, is that the subdomain solves are presumed to be carried out exactly. In the general nonseparable case of interest, fast Poisson solvers are not available, and one is faced with the necessity of direct sparse solvers in the computation of Cp^k, or of nested iterations. In the latter case, the convergence criterion for the inner iterations could conceivably be tuned to the rate of progress of the outer iteration to save computational work, but a more fully coupled framework of simultaneous iteration is possible, which we review in the next subsection.

2.3. Partitioned Matrix Methods (PMM)

The partitioned matrix methods are realized by taking $\mathcal{A}$ to be A of (2.3) and selecting appropriate $\mathcal{B}$. The iterations occur on vectors of length n, rather than n_0. We begin with the following theorem, due to Eisenstat [15].

Theorem 2.1. *Let the $n \times n$ partitioned matrix A and the $n_0 \times n_0$ Schur complement matrix C be defined as in (2.3) and (2.4), respectively, with corresponding right-hand sides f and g.*
(i) Algorithm PCG *applied to $Cv = g$ with initial iterate v^0 and preconditioner M is equivalent to algorithm* PCG *applied to $Au = f$ with initial iterate*

$$u^0 = \begin{pmatrix} v^0 \\ A_{11}^{-1}(f_1 - A_{10}v^0) \end{pmatrix}$$

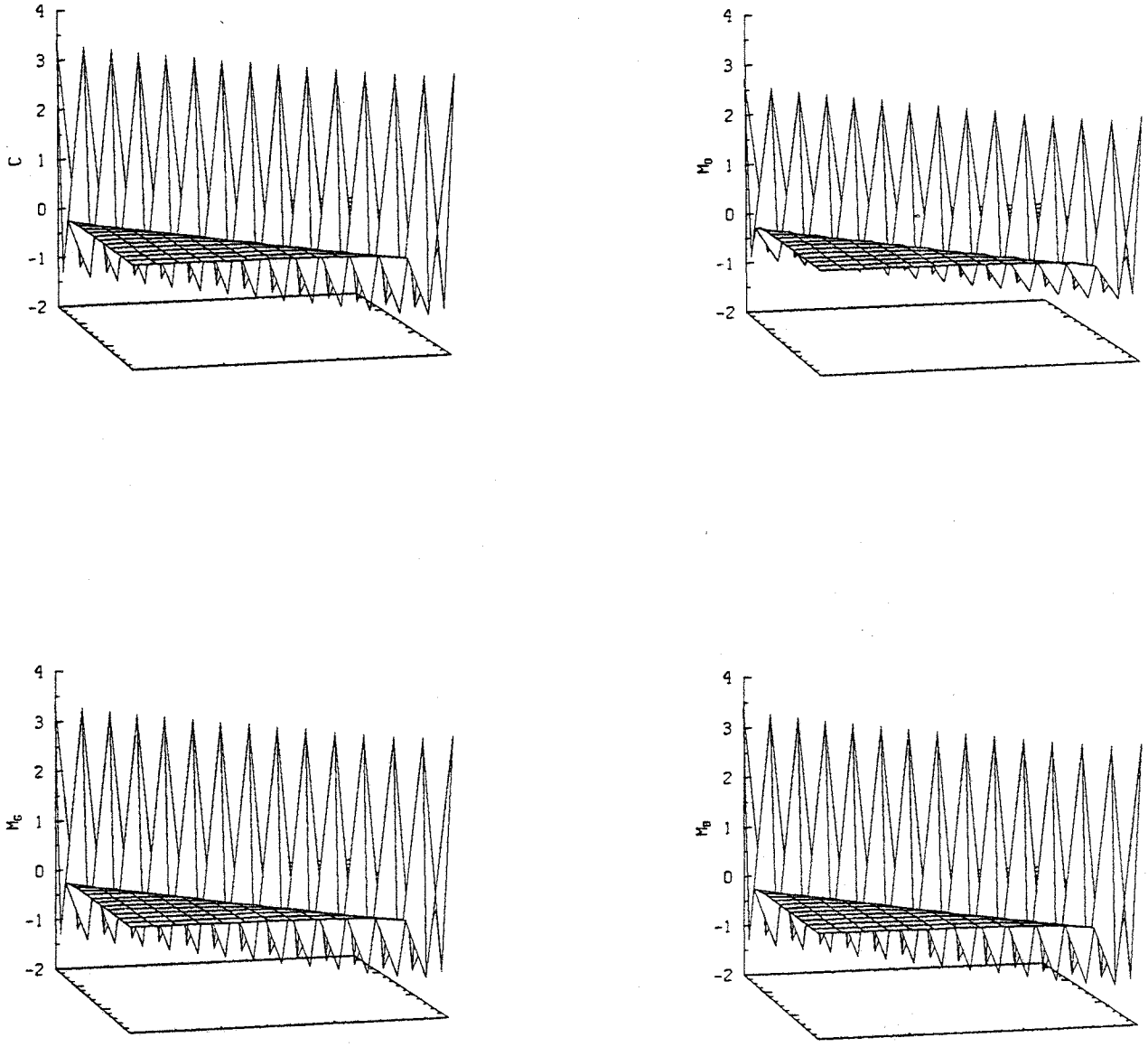


Figure 3: Surface plots of the elements of the Schur complement matrix C for Poisson's equation on a square uniformly discretized with 16 subintervals on a side, and various preconditioners. (a) C itself, (b) M_D, (c) M_G, (d) $M_C (= M_B)$.

and preconditioner

$$B = \begin{bmatrix} M + A_{01}A_{11}^{-1}A_{10} & A_{01} \\ A_{10} & A_{11} \end{bmatrix}, \tag{2.6}$$

in the sense that, for all $k \geq 0$,

$$u^k = \begin{pmatrix} v^k \\ A_{11}^{-1}(f_1 - A_{10}v^k) \end{pmatrix}.$$

(ii) There is no advantage to choosing an initial iterate more general than u^0 *as above, in the sense that* $||u^k - u||_A \leq ||w^k - u||_A$ *, where* w^k *is the* k*th iterate generated by PCG from the initial iterate*

$$w^0 = u^0 + \begin{pmatrix} 0 \\ \xi^0 \end{pmatrix}.$$

Proof. (i) Since A_{11}, M, and B are symmetric positive definite matrices, we may write $M = Q^TQ$ and $B = P^TP$, where

$$P = \begin{bmatrix} Q & 0 \\ A_{11}^{-1/2}A_{10} & A_{11}^{1/2} \end{bmatrix}.$$

The Schur complement system PCG iteration is equivalent to conjugate gradient (CG) iteration on $\hat{C}\hat{v} = \hat{g}$ with initial iterate $\hat{v}^{(0)} = 0$, where $\hat{C} \equiv Q^{-T}CQ^{-1}$, $\hat{v} \equiv Q(v - v^{(0)})$ and $\hat{g} \equiv Q^{-T}(g - Cv^{(0)})$; and similarly the partitioned system PCG iteration is equivalent to CG iteration on $\hat{A}\hat{u} = \hat{f}$ with initial iterate $\hat{u}^{(0)} = 0$, where $\hat{A} \equiv P^{-T}AP^{-1}$, $\hat{u} \equiv P(u - u^{(0)})$ and $\hat{f} \equiv P^{-T}(f - Au^{(0)})$. But note that

$$\hat{A} = \begin{bmatrix} Q^T & A_{10}^TA_{11}^{-1/2} \\ 0 & A_{11}^{1/2} \end{bmatrix}^{-1} \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \begin{bmatrix} Q & 0 \\ A_{11}^{-1/2}A_{10} & A_{11}^{1/2} \end{bmatrix}^{-1} =$$

$$\begin{bmatrix} Q^{-T} & -Q^{-T}A_{10}^TA_{11}^{-1} \\ 0 & A_{11}^{-1/2} \end{bmatrix} \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \begin{bmatrix} Q^{-1} & 0 \\ -A_{11}^{-1}A_{10}Q^{-1} & A_{11}^{-1/2} \end{bmatrix} =$$

$$\begin{bmatrix} Q^{-T}(A_{00} - A_{01}A_{11}^{-1}A_{10}) & 0 \\ A_{11}^{-1/2}A_{10} & A_{11}^{1/2} \end{bmatrix} \begin{bmatrix} Q^{-1} & 0 \\ -A_{11}^{-1}A_{10}Q^{-1} & A_{11}^{-1/2} \end{bmatrix} = \begin{bmatrix} Q^{-T}CQ^{-1} & 0 \\ 0 & I \end{bmatrix} = \begin{bmatrix} \hat{C} & 0 \\ 0 & I \end{bmatrix}.$$

Similarly,

$$\hat{f} = \begin{bmatrix} Q^T & A_{10}^TA_{11}^{-1/2} \\ 0 & A_{11}^{1/2} \end{bmatrix}^{-1} \begin{pmatrix} f_0 - A_{01}A_{11}^{-1}(f_1 - A_{10}v^0) - A_{00}v^0 \\ 0 \end{pmatrix} =$$

$$\begin{bmatrix} Q^{-T} & -Q^{-T}A_{10}^TA_{11}^{-1} \\ 0 & A_{11}^{-1/2} \end{bmatrix} \begin{pmatrix} g - Cv^0 \\ 0 \end{pmatrix} = \begin{pmatrix} \hat{g} \\ 0 \end{pmatrix}.$$

Thus the partitioned matrix PCG iteration is equivalent to CG iteration on

$$\begin{bmatrix} \hat{C} & 0 \\ 0 & I \end{bmatrix} \begin{pmatrix} \hat{v} \\ \xi \end{pmatrix} = \begin{pmatrix} \hat{g} \\ 0 \end{pmatrix}$$

with initial iterate $\xi^{(0)} = 0$, $\hat{v}^{(0)} = 0$ and the equivalence is established.

(ii) To prove optimality, we first define $\hat{w}^k \equiv P(w^k - u^0)$, and note that

$$\hat{w}^0 = \begin{pmatrix} 0 \\ A_{11}^{1/2}\xi^0 \end{pmatrix} , \quad \text{and } \hat{u} = \begin{pmatrix} \hat{v} \\ 0 \end{pmatrix} .$$

Then

$$\begin{aligned}
\|w^k - u\|_A^2 &= \|\hat{w}^k - \hat{u}\|_{\hat{A}}^2 \\
&= \min_{P_k} \|(I - \hat{A}P_k(\hat{A}))(\hat{w}^0 - \hat{u})\|_{\hat{A}}^2 \\
&= \min_{P_k} \left\| \begin{pmatrix} -(I - \hat{C}P_k(\hat{C}))\hat{v} \\ (I - P_k(I))A_{11}^{1/2}\xi^0 \end{pmatrix} \right\|_{\hat{A}}^2 \\
&= \min_{P_k} \left\{ \|(I - \hat{C}P_k(\hat{C}))\hat{v}\|_{\hat{C}}^2 + \|(1 - P_k(1))A_{11}^{1/2}\xi^0\|_2^2 \right\} \\
&\geq \min_{P_k} \left\{ \|(I - \hat{C}P_k(\hat{C}))\hat{v}\|_{\hat{C}}^2 \right\} \\
&= \min_{P_k} \|(I - \hat{A}P_k(\hat{A}))(\hat{u}^0 - \hat{u})\|_{\hat{A}}^2 \\
&= \|\hat{u}^k - \hat{u}\|_{\hat{A}}^2 \\
&= \|u^k - u\|_A^2,
\end{aligned}$$

where P_k is the set of polynomials of order k, and the second and penultimate steps rely on the optimality property of CG iteration (see [8], chapter 3, for instance).

∎

By this theorem, any Schur complement system preconditioner M can be applied to the interfacial equations of the partitioned matrix by its incorporation into the matrix B as shown in (2.6). Moreover, the theorem suggests a form for the preconditioner for a more general algorithm in which is it not required that the subdomain solves corresponding to A_{11}^{-1} be carried out exactly. For instance, if A derives from a nonseparable operator L, fast Poisson solvers may be used to precondition the subdomain solves. More costly exact subdomain solves are thereby avoided. To allow for this type of generality, we take $\mathcal{B}$ to be $\tilde{B}$, where

$$\tilde{B} = \begin{bmatrix} \tilde{B}_{00} & \tilde{B}_{01} \\ \tilde{B}_{10} & \tilde{B}_{11} \end{bmatrix} \equiv \begin{bmatrix} M + \tilde{A}_{01}\tilde{A}_{11}^{-1}\tilde{A}_{10} & \tilde{A}_{01} \\ \tilde{A}_{10} & \tilde{A}_{11} \end{bmatrix}, \tag{2.7}$$

and where the $\tilde{A}_{ij}$ are determined on a problem-specific basis. A convenient form for the $\tilde{A}_{ij}$ may be derived from a permutation conformal to that of (2.3) of the discrete Galerkin equations for

$$\tilde{A}_\Omega(u, v) = (f, v), \tag{2.8}$$

where

$$\tilde{A}_\Omega(u, v) \equiv \sum_{k=1}^{2} \tilde{A}_{\Omega_k}(u, v),$$

where for each subdomain

$$\tilde{A}_{\Omega_k}(u, v) \equiv \sum_{p,q=1}^{2} \int_\Omega \tilde{a}_{pq}^k \frac{\partial u}{\partial x_p} \frac{\partial v}{\partial x_q} \, \mathrm{dx},$$

and where the constants $\tilde{a}_{pq}^k$ are chosen for each subdomain k in such a way that the resulting matrix $\tilde{A}$ is spectrally equivalent to A. This iteration is, of course, no longer equivalent to any

reduced system iteration in the manner described above, and the scaling of M relative to the $\tilde{A}_{ij}$ becomes an issue, where it was not previously.

An efficient implementation of partitioned matrix preconditioners of the form (2.7) has been presented by Bramble, Pasciak & Schatz [4], wherein it is described how the operation $\mathcal{B}^{-1}r$ can be carried out at the cost of *two* subdomain solves per subdomain per iteration, plus the cost of applying M^{-1}. In the language of function-space decompositions, the solution u is written as the sum of a harmonic component, u^H, and a perpendicular component, u^P, such that in each subdomain k, $u^P|_k \in H^1_{0kh}$ satisfies

$$\tilde{A}_{\Omega_k}(u^P, v) = \tilde{A}_{\Omega_k}(u, v)$$

for all $v \in H^1_{0kh}$; and $u^H|_k \in H^1_{kh}$ satisfies $u^H = u$ on γ_{12} and

$$\tilde{A}_{\Omega_k}(u^H, v) = 0$$

for all $v \in H^1_{0kh}$.

In matrix terms, to solve

$$\tilde{B}\begin{pmatrix} u_0 \\ u_1 \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \end{pmatrix}$$

for u, one first solves

$$\tilde{A}_{11}u_1^P = f_1$$

for u_1^P, then

$$Mu_0 = f_0 - \tilde{A}_{01}u_1^P$$

for u_0, then

$$\tilde{A}_{11}u_1^H = -\tilde{A}_{10}u_0$$

for u_1^H and sets $u_1 = u_1^H + u_1^P$.

It is tempting to consider as a more economical alternative for $\mathcal{B}$ the matrix

$$\begin{bmatrix} M & 0 \\ \tilde{A}_{10} & \tilde{A}_{11} \end{bmatrix},$$

whose inversion allows skipping the first of the three steps above (and thus leads to an algorithm with no more subdomain solves per iteration than SCM), but this is not a symmetric preconditioner. The penalty for preserving a nest-free iterative structure in the case where the subdomain solves with A_{11} are too expensive to be done directly is thus an extra solve per subdomain with $\tilde{A}_{11}$ in the preconditioning step, and also the extra work in the multiplication and dot product steps due to the vector length of n instead of n_0.

In [5] the same authors consider a generalization of the Bjorstad-Widlund preconditioning in which the M of (2.7) is given by

$$\tilde{C}^{(1)} = \tilde{A}^{(1)}_{00} - \tilde{A}^{(1)}_{01}(\tilde{A}^{(1)}_{11})^{-1}\tilde{A}^{(1)}_{10}.$$

In matrix terms, to solve

$$\tilde{B}\begin{pmatrix} u_0 \\ u_1 \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \end{pmatrix}$$

for u, one first solves the Dirichlet problem in Ω_2:

$$\tilde{A}_{11}^{(2)} v = f_1^{(2)}$$

then the Neumann problem on Ω_1:

$$\begin{bmatrix} \tilde{A}_{00}^{(1)} & \tilde{A}_{01}^{(1)} \\ \tilde{A}_{10}^{(1)} & \tilde{A}_{11}^{(1)} \end{bmatrix} \begin{pmatrix} u_0 \\ u_1^{(1)} \end{pmatrix} = \begin{pmatrix} f_0 - \tilde{A}_{01}^{(2)} v \\ f_1^{(1)} \end{pmatrix}$$

then the Dirichlet problem in Ω_2:

$$\tilde{A}_{11}^{(2)} u_1^{(2)} = f_1^{(2)} - \tilde{A}_{10}^{(2)} u_0.$$

Since one of the subdomains solves is eliminated, the serial complexity of this method is less than that of the function-space decomposition technique just described, but the remaining subdomain solves are inherently sequential, and generalization to the many-subdomain case can only be carried out for decompositions (such as stripwise) in which the boundaries of the subdomains on which the natural boundary conditions are posed do not intersect.

2.4. Modified Schur Complement (MSC) preconditioners

Because the Schur complement matrix C is close to being tridiagonal, as illustrated for the Laplacian operator in Figure 3, it is natural to consider the use of tridiagonal or other low-bandwidth preconditioners for it. Such an approximation of one matrix by another constrained to satisfy various sparsity requirements has often proved useful in connection with iterative methods. We can generate a class of interfacial preconditioners of the form

$$M_{S(k)} = \tilde{A}_{00} - E_k, \tag{2.9a}$$

where E_k is a symmetric matrix with semi-bandwidth k which satisfies

$$E_k v_i = (\tilde{A}_{01} \tilde{A}_{11}^{-1} \tilde{A}_{10}) v_i \tag{2.9b}$$

for some $k+1$ vectors v_i, $i = 0, \ldots, k$. We have used $\tilde{A}_{11}$ rather than A_{11} in (2.9b) because this technique is motivated by variable coefficient problems for which exact subdomain solvers are too expensive to consider.

Note that the construction of E_k requires $k+1$ solves on each subdomain, which is in general $(k+1)/2$ times the cost of one preconditioning step with $\mathcal{B}$ of the form (2.7). For $k = 0$, we consider the vector $v_0 = (1, 1, \ldots)^T$; for $k = 1$, the vectors $v_0 = (1, 0, 1, 0, \ldots)^T$ and $v_1 = (0, 1, 0, 1, \ldots)^T$; and so forth. Equation (2.9b) gives $(k+1)n_0$ scalar equations for the $(k+1)(n_0 - k/2)$ distinct elements of E_k, but the overdetermination is consistent due to the symmetry of the product of matrices on the right-hand side. For the set of v_i recommended above, the diagonal elements of E_k can be read off from (2.9b) and the remaining elements can be obtained in $O(kn_0)$ operations. Note that $M_{S(k)}^{-1}$ can also be applied at the cost of only $O(kn_0)$ operations once its factorization is stored.

Surface plots of $M_{S(0)}$, $M_{S(1)}$, $M_{S(2)}$ for the same model problem described in conjunction with Figure 3 are shown in Figure 4, and their associated estimated condition numbers also appear in Table 1.

For small k, which is the only practical limit, this class of preconditioners turns out to be markedly inferior to the optimal class described earlier for constant coefficient operators on uniform grids. However, it can be competitive when the interfaces are not placed along level curves of the coefficients. Figure 5 shows surface plots of the first three members of the $M_{S(k)}$ class and the Schur complement matrix itself for a problem with the same model geometry, but with the nonseparable

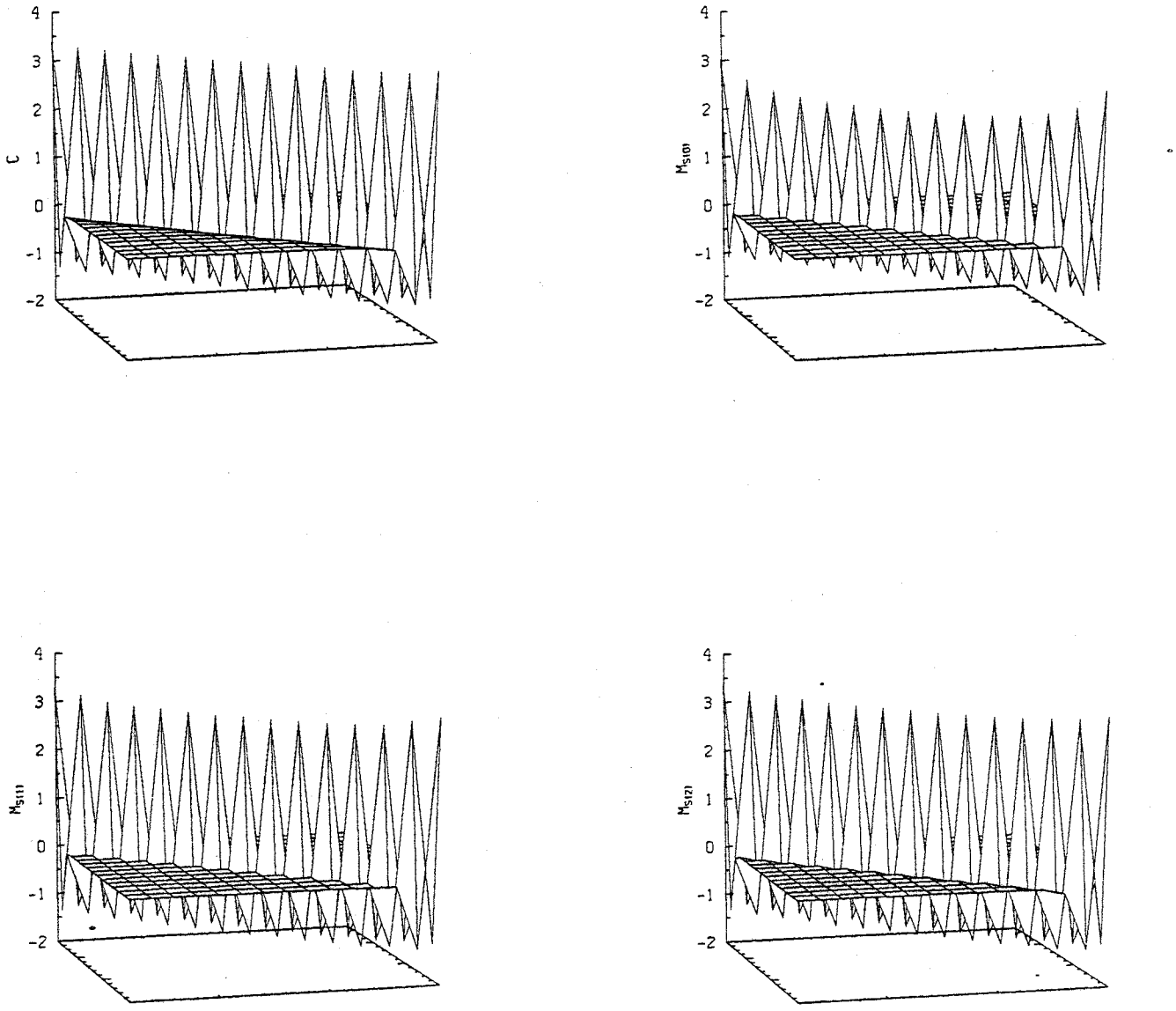


Figure 4: Surface plots of the elements of the Schur complement matrix C for Poisson's equation on a square uniformly discretized with 16 subintervals on a side, and the first three MSC preconditioners. (a) C itself, (b) $M_{S(0)}$, (c) $M_{S(1)}$, (d) $M_{S(2)}$.

operator equation $\nabla \cdot (a \nabla u) = f$, where $a = 1 + b \tan^{-1}(x - \frac{1}{2}) + c \tan^{-1}(d(y - \frac{1}{2}))$, and with all of the tilde-quantities in (2.9) obtained from the subdomain averaging of the x- and y-direction diffusion coefficients in the opposite direction to achieve separability within subdomains. Estimated condition numbers for this problem, using PMM with interfacial preconditioners from both the optimal and MSC classes appear in Table 7 of §3.2. Observe the sensitivity of the MSC preconditioners to the variation of a along the interface, which is lacking in the optimal preconditioners. The plots are for $b = 0.65$, $c = 0.35$, $d = 10.0$.

The MSC preconditioners have the advantage of being self-scaling, relative to the $\tilde{A}_{ij}$, and can enjoy a decisive advantage over a poorly scaled optimal interface preconditioner in the PMM context. However, it is not difficult to choose a good scaling for the optimal preconditioners (see [4]).

2.5. Methods of Variational Type

Glowinski and several coauthors have contributed extensively to the literature of domain decomposition techniques in the course of their work on the numerical modeling of steady, compressible inviscid flow and of unsteady, incompressible viscous flow (see [16] and the references therein). Both of these subrealms of the Navier-Stokes equations may be operator-split and discretized in such a way that the subtasks of greatest computational complexity are scalar problems of Poisson type with a large number of degrees of freedom and wide variability in the coefficients in irregular geometry. We briefly describe here three algorithms of conjugate gradient type presented in [10, 16] which enter into the comparisons in §3. Like the Schur complement methods, all of these methods iterate on degrees of freedom at the boundaries of the subdomains only, with each iteration requiring one or two exact interior solves per subdomain. Two of the methods use a partitioning of the domain into subdomains with non-intersecting interiors, as in Figures 1 and 2. The remaining method is related to the Schwartz alternating procedure in that the subdomains overlap in regions of nonzero measure.

The algorithms are summarized below in matrix operator form, without derivation, for the model Poisson problem (2.1b). It suffices to identify the vectors in Algorithm PCG in terms of the physical variables and to specify the operators $\mathcal{A}$ and $\mathcal{B}^{-1}$.

The first two algorithms are based on the equivalence of

$$\begin{aligned} -\Delta y &= f \text{ in } \Omega, \\ y &= 0 \text{ on } \Gamma, \end{aligned}$$

and the problem

$$y = \arg \min_{z \in H_0^1(\Omega)} \{ \frac{1}{2} \int_\Omega | \nabla z |^2 \ \mathrm{dx} - \int_\Omega f z \ \mathrm{dx} \}.$$

Upon partitioning the domain and carrying out the minimization on each of the subdomains separately, either the normal derivative or the trace of the solution along the interface can be regarded as the principal unknown, and its value iteratively updated until continuity of the other is satisfied. Through intermediate saddle-point formulations dual conjugate gradient algorithms are derived.

In the first algorithm (§4.1 of [10] or §2.2.2 of [16]), the unknown vector represents the discrete normal derivative of the solution along the interface adjusted for sign, $x = (-1)^{k-1} \partial u_k / \partial n_k$, $\mathcal{A}$ is the matrix $(C^{(1)})^{-1} + (C^{(2)})^{-1}$, and $\mathcal{B}$ is the matrix defined by (for $k = 1$ or $k = 2$)

$$[\mathcal{B}]_{ij} = \tilde{h}^{-1} \int_{\gamma_{12}} \psi_{kj} \psi_{ki} \ \mathrm{ds}, \ i, j \in N_0.$$

In the uniform mesh case, $\mathcal{B} = I - \frac{1}{6} K$. This method we denote by Saddle-1.

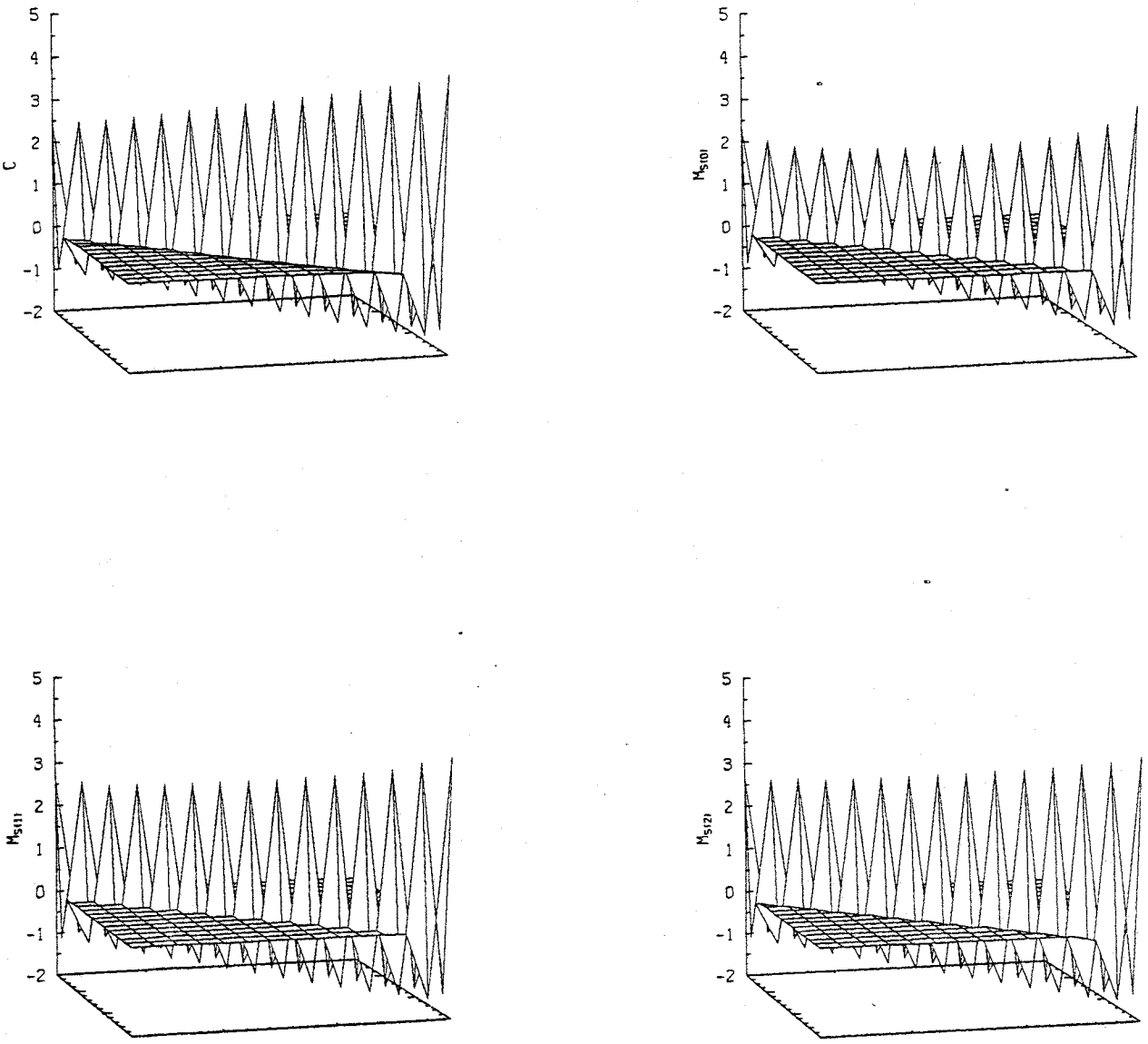


Figure 5: Surface plots of the elements of the Schur complement matrix C for a nonseparable operator on a square uniformly discretized with 16 subintervals on a side, and the first three MSC preconditioners. (a) C itself, (b) $M_{S(0)}$, (c) $M_{S(1)}$, (d) $M_{S(2)}$.

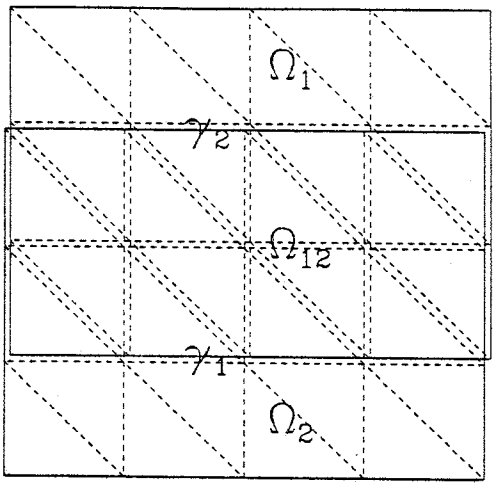


Figure 6: Sample domain illustrating the overlap decomposition (subdomains staggered for clarity).

In the second algorithm (§4.2 of [10]), denoted Saddle-2, the unknown vector represents the solution itself at the interface nodes, $\mathcal{A}$ is precisely the Schur complement C, and $\mathcal{B}$ is the matrix defined by

$$[\mathcal{B}]_{ij} = \sum_{k=1}^{2} \int_{\Omega_k} \nabla\psi_{kj} \cdot \nabla\psi_{ki} \, \mathrm{dx}, \; i,j \in N_0.$$

In the uniform mesh case $\mathcal{B} = 2(I + \frac{1}{2}K)$. These preconditioners commute with their respective $\mathcal{A}$-matrices, the spectra of which can be constructed from the eigendecompositions given in §§2.2. It is readily verified that in both cases $\kappa(\mathcal{B}^{-1}\mathcal{A})$ grows asymptotically in proportion to $1/h$.

The other method (§2.3.5 and §2.3.7 of [16]) uses a decomposition with overlapping subdomains, as illustrated for a two subdomain case in Figure 6. The overlap region is denoted by Ω_{12} and distinct internal subdomain boundaries γ_1 and γ_2 are defined as shown. We denote this method by CG-Schwarz (or CGS).

The unknown vector represents the trace of the solution along $\gamma_1 \cup \gamma_2$, and is partitioned accordingly, $u \equiv (u_1, u_2)^T$. For the case of the Laplacian, the algorithm is based on the equivalence of

$$\begin{aligned} -\Delta y &= f \text{ in } \Omega, \\ y &= 0 \text{ on } \Gamma, \end{aligned}$$

and the problem

$$u \equiv (u_1, u_2) = \arg \min_{v \in V_1 \times V_2} \{\frac{1}{2} \int_\Omega \left[|\nabla(y_2 - y_1)|^2 + |y_2 - y_1|^2 \right] \mathrm{dx}\},$$

where the $y_k \in H^1(\Omega_k)$ are the solutions of

$$\begin{aligned} -\Delta y_k &= f_k \text{ in } \Omega_k, \\ y_k &= 0 \text{ on } \partial\Omega_k \cap \Gamma, \\ y_k &= v_k \text{ on } \gamma_k, \end{aligned}$$

and where V_k is the space consisting of the traces on γ_k of functions in $H^1(\Omega_k)$ which vanish on $\partial\Omega_{12} \cap \Gamma$, for $k = 1, 2$.

We introduce H^1_{12h}, the discrete subspace of $H^1(\Omega_{12})$ consisting of piecewise linear functions vanishing on $\partial\Omega_{12} \cap \Gamma$, but not necessarily on γ_1 or γ_2, and we denote a basis for it by $\{\chi_j\}, j \in N_{12}$. Let N_{01} and N_{02} be index sets for the nodes along γ_1 and γ_2, respectively.

The action of $\mathcal{A}$ on a vector $x \equiv (x_1, x_2)^T$ is implicitly defined as follows. First, the pair of subdomain Dirichlet problems with boundary data x_k on the γ_k,

$$A_k y_k = -C_k x_k,$$

is solved for the y_k, where

$$[A_k]_{ij} = \int_{\Omega_k} \nabla\psi_{kj} \cdot \nabla\psi_{ki} \, \mathrm{dx}, \; i, j \in N_k$$

and

$$[C_k]_{ij} = \int_{\Omega_k} \nabla\psi_{kj} \cdot \nabla\psi_{ki} \, \mathrm{dx}, \; i \in N_k, j \in N_{0k}.$$

Then the difference of the y_k over the region of common definition,

$$\delta y_k = y_k|_{\Omega_{12}} - y_l|_{\Omega_{12}},$$

is computed, where l is the complement of k in $\{1, 2\}$. Then another pair of subdomain Dirichlet problems with forcing due to the difference in the y_k over the overlap region,

$$A_k z_k = F_k \delta y_k,$$

is solved for the z_k, where

$$[F_k]_{ij} = \int_{\Omega_{12}} [\nabla\chi_j \cdot \nabla\psi_{ki} + \chi_j \psi_{ki}] \, \mathrm{dx}, \; i \in N_k, j \in N_{12}.$$

Finally,

$$\mathcal{A}x = \begin{pmatrix} D_1 \delta y_1 - E_1 z_1 \\ D_2 \delta y_2 - E_2 z_2 \end{pmatrix},$$

where

$$[D_k]_{ij} = \int_{\Omega_{12}} [\nabla\chi_j \cdot \nabla\psi_{ki} + \chi_j \psi_{ki}] \, \mathrm{dx}, \; i \in N_{0k}, j \in N_{12}$$

and

$$[E_k]_{ij} = \int_{\Omega_k} \nabla\psi_{kj} \cdot \nabla\psi_{ki} \, \mathrm{dx}, \; i \in N_{0k}, j \in N_k.$$

We may consider different preconditioners, which are of the form $\mathcal{B} = diag(\mathcal{B}_1, \mathcal{B}_2)$. In §2.3.5 of [16], an essentially unpreconditioned form of the algorithm is proposed in which $\mathcal{B}_k$ has the interfacial line integral form of $\mathcal{B}$ in the Saddle-1 method. In the experiments in §3 we in fact report on $\mathcal{B}_k = I$. In §2.3.7 the preconditioner

$$[\mathcal{B}_k]_{ij} = \int_{\Omega_k} \nabla\psi_{kj} \cdot \nabla\psi_{ki} \, \mathrm{dx}, \; i, j \in N_{0k}$$

is used. Though not mentioned in [16], we also consider $\mathcal{B}_k = K^{1/2}$ in the numerical experiments. Note that, like the PMM, the overlap decomposition-based method requires two Dirichlet problems to be solved in each subdomain per iteration.

2.6. Generalization to Multiple Nonintersecting Interfaces

The two-subdomain case may possess genuine interest for many problems of physical origin, but it is of limited interest when implementation of domain decomposition algorithms on ensemble architectures with a large number of processors is considered. The multiple subdomain case has been studied by Dryja and Proskurowski in [11] and [12], in which serial implementations making use of up to 32 subdomains have been tested, and by Chan and Resasco in [7]. The multiple strip case can be accommodated by our notation of (2.3) and (2.4) by letting u_0 represent all of the separator set unknowns, ordered by interface, and letting u_1 represent all of the interior unknowns, ordered by subdomain. If Ω is uniformly triangulated, and if there are p subdomains of equal size with n_0 interior gridpoints along each interface and m gridpoints across, then A_{00} is a block matrix with $p-1$ diagonal blocks of size n_0, and A_{11} is a block matrix with p diagonal blocks of size $n_0 m$. Under our assumptions on L, the Schur complement (which is block tridiagonal with blocks of size n_0) is symmetric and positive definite, and the following is proved in [11]:

Theorem 2.2. *Let the Schur complement C be defined as following (2.4) and above, and let $\tilde{M}_D$ be defined as the $(p-1)\times(p-1)$ block diagonal matrix with $n_0 \times n_0$ diagonal blocks M_D. Then, for all x,*

$$w\gamma_0(x,\tilde{M}_D x) \le (x,Cx) \le w^{-1}\gamma_1(x,\tilde{M}_D x)\ , \tag{2.10}$$

where γ_0 and γ_1 are positive constants independent of h, and w is the minimum of the strip widths.

This makes $\tilde{M}_D$ an optimal preconditioner for C for a given decomposition, and similar block-diagonal extensions can be made for the other single-interface preconditioners. The analogous block preconditioner $\tilde{M}_G$ is also tested in [11]. The reference [12] considers a multiple subdomain generalization of the alternating Neumann-Dirichlet method, first presented for the two-strip case in [2]. Finally, [7] presents an exact eigendecomposition of the multiple-strip Schur complement matrix which generalizes Chan's method [6] to a multiple-strip domain-decomposed fast Poisson solver.

Note from (2.10) that the block preconditioning of C by $\tilde{M}_D$ suffers as the strips become thin, that is as the number of subdomains is increased for fixed domain geometry. In [7] this is traced to ignoring the off-diagonal blocks of C. By means of the exact eigendecomposition, these blocks may be shown to be insignificant in the limit of large aspect ratio subdomains ("thick" strips), but more and more significant in the limit of small aspect ratio subdomains. Since $w \propto p^{-1}$, the bound on the condition number from (2.10), $\gamma_1/(\gamma_0 w^2)$, increases like p^2. The inability of stripwise decompositions to accommodate large numbers of subdomains without a deterioration of convergence is a major weakness when it comes to large-scale parallel implementations. A more implicit means of handling the interfaces is required.

2.7. Generalization to Intersecting Interfaces

An extension to decompositions in which interfaces intersect in *vertices* or *crosspoints*, resulting in the formation of boxes instead of strips, is given by Bramble *et al* in [4]; this development is in fact the focus of their paper. (The single interface version of the PMM presented in §§2.3 is a special case.) Constraints of space and focus prohibit the full development of their technique here, but a brief account of its implications for the discrete version of the problem is furnished below.

The key step is the further decomposition in function space of the discrete harmonic component of u, u^H. Recall from §§2.3 that $u = u^P + u^H$, where in each subdomain k, $u^P|_k$ vanishes on $\partial\Omega_k$ and satisfies

$$\tilde{A}_{\Omega_k}(u^P, v) = \tilde{A}_{\Omega_k}(u, v)$$

for all $v \in H^1_{0kh}$; and $u^H|_k$ satisfies $u^H = u$ on $\partial\Omega_k$ and

$$\tilde{A}_{\Omega_k}(u^H, v) = 0$$

for all $v \in H^1_{0kh}$. We now set $u^H = u^E + u^V$, where in each subdomain k, $u^V|_k$ and $u^E|_k$ each satisfy

$$\tilde{A}_{\Omega_k}(u^V, v) = \tilde{A}_{\Omega_k}(u^E, v) = 0$$

for all $v \in H^1_{0kh}$, and u^V is linear along the edges of $\partial\Omega_k$ and and agrees with u at the vertices, and u^E vanishes at the vertices. A finite element basis for u is then constructed which consists of the usual C^0 piecewise linear basis functions of smallest possible support defined at the nodes interior to each Ω_k and at the nodes along each edge (except at the intersections of the edges), and a special set of basis functions defined at the crosspoints with support extending out along the edges which connect the crosspoints. These special basis functions each vanish at all crosspoints but one and at all interior nodes, and are linear along the edges. The Galerkin equations involving the crosspoint degrees of freedom thus possess a nonlocal connectedness. If the nodal values of u are permuted as before so that u_0 represents the separator nodes (edges *and* crosspoints), and u_1 the interior nodes, then the computation of u_1^P and u_1^H proceed as before, as independent Dirichlet solves over each subdomain. However, the calculation of u_0 from

$$Mu_0 = f_0 - \tilde{A}_{01}u_1^P$$

proceeds in another series of independent solves as follows. M is a block diagonal matrix with a diagonal block corresponding to the nodes along each edge. These blocks are each essentially matrices of the form M_D, leaving scaling considerations aside. The final block corresponds to a diagonally dominant difference equation for the internal crosspoint nodes and has a sparsity structure identical to a graph of the decomposition: each edge connecting one vertex i to another vertex j contributes a nonzero to the ith row in column j. For a decomposition of the unit square into uniform square subdomains, this matrix is simply the discrete Laplacian, to within a scaling factor. The right-hand side involves inner products between the interfacial nodal coefficient vectors. The reader is referred to [4] for the complete details. We conclude by paraphrasing Theorem 1 and Remark 2.6 therein as follows, in which it is assumed that the underlying triangulation and decomposition into subdomains are quasi-uniform of sizes h and d, respectively.

Theorem 2.3. *Let the matrix A be defined as in (2.3) and above, and let B be defined as in (2.7), where M is the separator set preconditioner of [4]. Then, for all x,*

$$\gamma_0(x, Bx) \leq (x, Ax) \leq \gamma_1(x, Bx) \ , \tag{2.11}$$

where γ_0 and γ_1 are positive constants such that for some positive constant c_1,

$$\frac{\gamma_1}{\gamma_0} \leq c_1(1 + \ln(\frac{d}{h})^2) \ .$$

If, instead, the vertex difference equation block of M is replaced by a weighted identity operator then for some positive constant c_2,

$$\frac{\gamma_1}{\gamma_0} \leq c_2 d^{-2}(1 + \ln(\frac{d}{h})^2) \ .$$

The implications of this theorem for the many subdomain decomposition are illustrated by the following special case. Consider a square domain uniformly discretized with n subintervals on a side and uniformly decomposed into p square subdomains of $n/\sqrt{p}$ subintervals on a side. Then $d \propto \sqrt{p}$ and $d/h = n/\sqrt{p}$. Therefore, by Theorem 2.3, when the crosspoints are treated implicitly the condition number is bounded by $c_1(1+\ln(n/\sqrt{p})^2)$; and when they are decoupled the condition number is bounded by a $c_2 p(1+\ln(n/\sqrt{p})^2)$. In the former case the condition number is bounded by a constant if subdomains are introduced as required with increasing resolution to keep the number of subintervals on a subdomain side fixed. In the latter case, the condition number grows in proportion to the number of subdomains.

A simpler alternative for handling the crosspoints in the decoupled case is to employ the usual cardinal finite element basis for all of the interior degrees of freedom, including the crosspoints, along with a weighted identity operator for the corresponding block of M. This method requires no special computations to form the right-hand side of the crosspoint system. The examples of §3 labeled "PMM without vertex coupling" employ this simpler alternative.

3. Experimental Comparisons of Domain Decomposition Techniques

The methods described in section 2 were tested on a variety of model problems chosen to reveal their relative strengths and weaknesses. Four of the problems we consider are posed on the unit square, symmetrically divided into two strips or four boxes by straight segments bisecting opposite sides. The operators include the cases of uniform, discontinuous, and smoothly varying nonseparable coefficients. The other problems are posed in a small aspect ratio rectangle, and in a T-shaped region.

These tests were carried out on a VAX/785 in single precision (24-bit mantissa) using the experimental interpretive language *CLAM* (Conversational Linear Algebra Machine). By providing a convenient symbolic interface to the LINPACK and EISPACK libraries, *CLAM* allowed relatively quick "breadboarding" of the various algorithms. The tests are not as comprehensive as some of those available in the literature for individual problems and methods (*e.g.*,[1, 4, 11, 12]) because of size, speed, and precision limitations, nor do they embrace as wide a scope of domain geometries or operators. The chief value of these results is tutorial, in that they permit algorithmic comparison on a common set of problems from common initial iterates with common convergence criteria and measures, all of which may vary or be left unstated from paper to paper.

3.1. Conventions in conducting tests and reporting results

To complete the specification of Algorithm PCG of §§2.1, which is the basis for all of the experiments, the initial iterate is always taken to be zero everywhere, and convergence is based on the relative size of the unnormalized Euclidean norm of the true residual. Since these practices are not uniform in the testing of such algorithms, we comment briefly upon them. In practical applications of PCG, the most conveniently monitored convergence criterion is the norm of the preconditioned residual, $(r, \mathcal{B}^{-1}r)^{\frac{1}{2}}$, since the square of this quantity is already required to advance the algorithm. However, when comparisons across a variety of preconditioners $\mathcal{B}$ are carried out, the convergence criterion should not be affected by the type of preconditioner. Therefore the norm of the unpreconditioned residual was calculated at every iteration of the serial tests, as well. We declare convergence when $(r^k, r^k)^{\frac{1}{2}} < \epsilon(r^0, r^0)^{\frac{1}{2}}$, as opposed to the absolute criterion used in some of the early domain decomposition literature, in order to remove dependence on the resolution of the problem. In our test problems the shape of the right-hand side f is generally such that its domain average decreases by an appreciable factor as the mesh is refined over several powers of two and points further from the symmetrically located maximum of f are brought in. This is reflected in a reduced initial residual, since the initial iterate is zero, and fewer iterations are required to reach an absolute residual tolerance. If anything, we should insist that an iterative method work

harder as the resolution increases in order to realize the potential of the lower truncation error of the discretization, otherwise the additional resolution is wasted from the continuous viewpoint. As a minimum requirement, we insist that the residual be decreased by a constant factor. In all of the tables to follow, $\epsilon = 10^{-4}$.

The principal results that we report for each experiment are the number of iterations to convergence, I, the estimated condition number of the preconditioned system, κ, and the average reduction per iteration of the residual, ρ. These are complementary indications of the rate of convergence of the method. As the only one of these three quantities which depends solely upon the operator, the true condition number is an attractive measure. However, a large condition number may lead to an overly pessimistic appraisal of a method if the eigenvalues are unevenly distributed. The iteration count is the most useful "bottom line" convergence measure for use in conjunction with cost-per-iteration complexity estimates to determine overall algorithmic complexity. However, this is convergence criterion-dependent. The average rate of reduction per iteration, defined by $((r^I, r^I)/(r^0, r^0))^{1/(2I)}$, where I is the number of iterations to convergence, is a more reliable estimator of the convergence rate than the condition number, without being as strongly dependent upon the particular convergence criterion as I, and it does not exhibit the threshold effect that I does. However, ρ still retains a dependence on the initial iterate. Since none of these three measures is ideal, we report all of them.

The condition number estimate in our tables is obtained as a by-product of the PCG iterations by the method of Lanczos [20] at a small computational overhead. For each k in the loop ($PCG.6$)-($PCG.14$), the diagonal and subdiagonal elements of a symmetric tridiagonal matrix are formed according to

$$d_k = \frac{1}{\alpha^k} + \frac{\beta^{k-1}}{\alpha^{k-1}} , \quad s_{k+1} = -\frac{\sqrt{\beta^k}}{\alpha^k} ,$$

respectively. The extreme eigenvalues of this matrix often accurately approximate those of $B^{-1}A$ even if terminated with k considerably less than n.

Finally, we note that although the finite element formulation has been chosen throughout this paper in order to provide a uniform theoretical background for the different methods, the results in Tables 5, 6, and 7 were actually generated using the standard second-order finite difference method to construct the discrete A and $\hat{A}$. (These alternative formulations do not generally coincide at the discrete level when the operator is not piecewise constant.)

3.2. Stripwise decomposition tests

The first test problem is Poisson's equation

$$\nabla^2 u = f ,$$

in the unit square divided symmetrically into two strips, where f is chosen so that $u = 16xy(1-x)(1-y)$. This problem is essentially the same as the first example in [11, 12]. It may be solved conveniently by a SCM. Surface plot comparisons of the preconditioners in Table 1 with C were given in Figure 3.

For this domain geometry, and for all but one to follow, the Schur complement matrix preconditioners M_C and M_B are identical, so there is no need for separate tabulation. The optimality properties of M_C, M_G, and M_D are in evidence, with the number of iterations independent of problem size. As the mesh is refined, ρ actually shows a slight improvement for M_G and M_D. For any meshes but the coarsest, the MSC preconditioners are inferior, and exhibit steadily deteriorating ρ. The last column, for which $B = I$, shows the proportionality of the condition number to the number of degrees of freedom in the unpreconditioned system.

h^{-1}		M_C	M_G	M_D	$M_{S(0)}$	$M_{S(1)}$	$M_{S(2)}$	I
8	I	1	2	3	4	3	2	4
	κ	1.000	1.094	1.257	1.320	1.154	1.045	6.317
	ρ	1.1(-6)	1.1(-3)	3.2(-2)	3.6(-3)	1.0(-3)	5.9(-3)	1.1(-2)
16	I	1	2	3	5	5	4	8
	κ	1.000	1.091	1.303	1.760	1.486	1.297	13.063
	ρ	3.7(-6)	9.0(-4)	3.2(-2)	1.5(-1)	8.8(-2)	4.2(-2)	1.1(-1)
32	I	1	2	3	7	6	6	12
	κ	1.000	1.091	1.320	2.472	2.066	1.756	26.06
	ρ	1.4(-5)	8.0(-4)	2.7(-2)	2.4(-1)	2.0(-1)	1.4(-1)	4.4(-1)
64	I	1	2	3	9	8	7	17
	κ	1.000	1.090	1.337	3.523	2.941	2.483	52.43
	ρ	8.4(-5)	8.0(-4)	2.2(-2)	3.3(-1)	3.0(-1)	2.5(-1)	5.7(-1)

Table 1: $\nabla^2 u = f$ on the unit square, divided symmetrically into two strips. Results for SCM as a function of problem size and interface preconditioner.

h^{-1}		M_C	M_G	M_D	$M_{S(0)}$	$M_{S(1)}$	$M_{S(2)}$
8	I	1	3	4	4	3	3
	κ	1.000	1.094	1.288	1.318	1.140	1.072
	ρ	1.1(-6)	3.8(-3)	7.4(-2)	3.2(-3)	2.7(-2)	4.0(-4)
16	I	1	3	4	5	4	4
	κ	1.000	1.091	1.346	1.760	1.484	1.290
	ρ	4.3(-6)	2.8(-3)	8.7(-2)	1.3(-1)	9.4(-2)	4.8(-2)
32	I	1	2	4	6	6	5
	κ	1.000	1.091	1.363	2.468	2.065	1.755
	ρ	9.0(-6)	9.0(-4)	9.1(-2)	2.0(-1)	1.6(-1)	1.2(-1)
64	I	1	2	4	7	7	6
	κ	1.000	1.091	1.370	3.276	2.933	2.479
	ρ	5.0(-5)	7.9(-3)	9.3(-2)	2.5(-1)	2.3(-1)	1.9(-1)

Table 2: $\nabla \cdot (a \nabla u) = 0$ on the unit square, divided symmetrically into two strips, where $a = 1.0$ in one subdomain and 0.1 in the other. Results for SCM as a function of problem size and interface preconditioner.

The second problem involves a physical interface which coincides with the subdomain boundary. The equation is:

$$\nabla \cdot (a \nabla u) = 0$$

where the diffusion coefficient a is 1.0 in one of the subdomains and 0.1 in the other, and where u obeys the Dirichlet condition $u = xy$ on the boundary of the unit square. Again, a SCM is appropriate for this problem.

The optimal methods continue to perform well in spite of the jump discontinuity, although ρ no longer shows improvement with increasing problem size. The MSC methods fare slightly better than in the featureless case, but still cannot compete with the optimal methods on so ideal a problem.

h^{-1}		M_C	M_B	M_G	M_D	$M_{S(0)}$	$M_{S(1)}$	$M_{S(2)}$
32	I	1	3	3	3	5	4	4
	κ	1.000	1.270	2.095	2.000	1.634	1.449	1.281
	ρ	7.6(-6)	1.6(-2)	2.2(-2)	3.3(-2)	1.1(-1)	9.4(-2)	6.6(-2)
64	I	1	3	3	3	6	6	5
	κ	1.000	1.271	2.089	2.047	2.243	1.963	1.639
	ρ	4.3(-5)	1.6(-2)	2.2(-2)	2.6(-2)	1.9(-1)	1.7(-1)	1.3(-1)

Table 3: $\nabla^2 u = f$ on a low aspect ratio rectangle, divided asymmetrically into two strips. Results for SCM as a function of problem size and interface preconditioner.

h^{-1}		M_C	M_G	M_D	$M_{S(0)}$	$M_{S(1)}$	$M_{S(2)}$
8	I	3	2	4	4	4	3
	κ	1.079	1.074	1.426	1.414	1.210	1.122
	ρ	9.5(-3)	9.9(-3)	9.3(-2)	7.4(-1)	5.2(-2)	2.4(-2)
16	I	3	3	5	5	5	4
	κ	1.111	1.109	1.468	1.718	1.599	1.380
	ρ	1.7(-2)	1.9(-2)	1.1(-1)	1.2(-1)	1.0(-1)	7.3(-2)
32	I	3	3	5	6	6	5
	κ	1.140	1.138	1.489	2.493	2.260	1.914
	ρ	3.3(-2)	3.3(-2)	1.1(-1)	2.2(-1)	1.9(-1)	1.5(-1)

Table 4: $\nabla^2 u = f$ on an asymmetric T-shaped region, divided into two strips. Results for SCM as a function of problem size and interface preconditioner.

The third problem is again Poisson's equation, a compressed version of the first problem posed in the rectangle $(0,1) \times (0, \frac{3}{8})$, with an interface at $y = \frac{1}{4}$ and f chosen so that $u = \frac{1024}{9} x(1-x)y(\frac{3}{8} - y)$.

The subdomain aspect ratios in this problem are more typical of those in multistrip decompositions of domains of roughly unit aspect ratio. This is the only problem to exhibit a distinction between M_C, which is again exact, and M_B.

As our only example of a geometry which is unsuitable for an undecomposed fast Poisson solver, we consider the asymmetric T-shaped domain problem from [1]. The domain is the unit square with two rectangles removed, as shown to scale in Figure 1b. The equation is Poisson's equation with inhomogeneous Dirichlet boundary data and right-hand side chosen so that $u = x^2 + y^2 - xe^x \cos y$. Though the overall geometry is nonseparable, the subdomains can each be handled by a fast Poisson solver, so a SCM is appropriate.

The optimal methods have no trouble handling geometrical irregularity such as this. The MSC methods are more competitive than before due to the spoiling of symmetry which prevents the use of any exact preconditioning, but they continue to be inferior for sufficiently fine resolution.

The fifth problem involves generalization in another direction: a nonseparable operator. We consider ELLPACK problem #1 [25]:

$$\nabla \cdot (\underline{a} \nabla u) + Ku = f \ ,$$

where $a_{11} = e^{xy}$, $a_{22} = e^{-xy}$, $a_{12} = a_{21} = 0$, $K = -(1 + x + y)^{-1}$, and where f is chosen so that $u = \frac{3}{4} e^{xy} \sin(\pi x) \sin(\pi y)$. Contour plots of a_{11}, a_{22} and K appear in Figure 7.

For this problem, we use a PMM and present results for two different forms of the partitioned matrix preconditioner $\mathcal{B}$ in two separate tables. The condition number estimates pertain

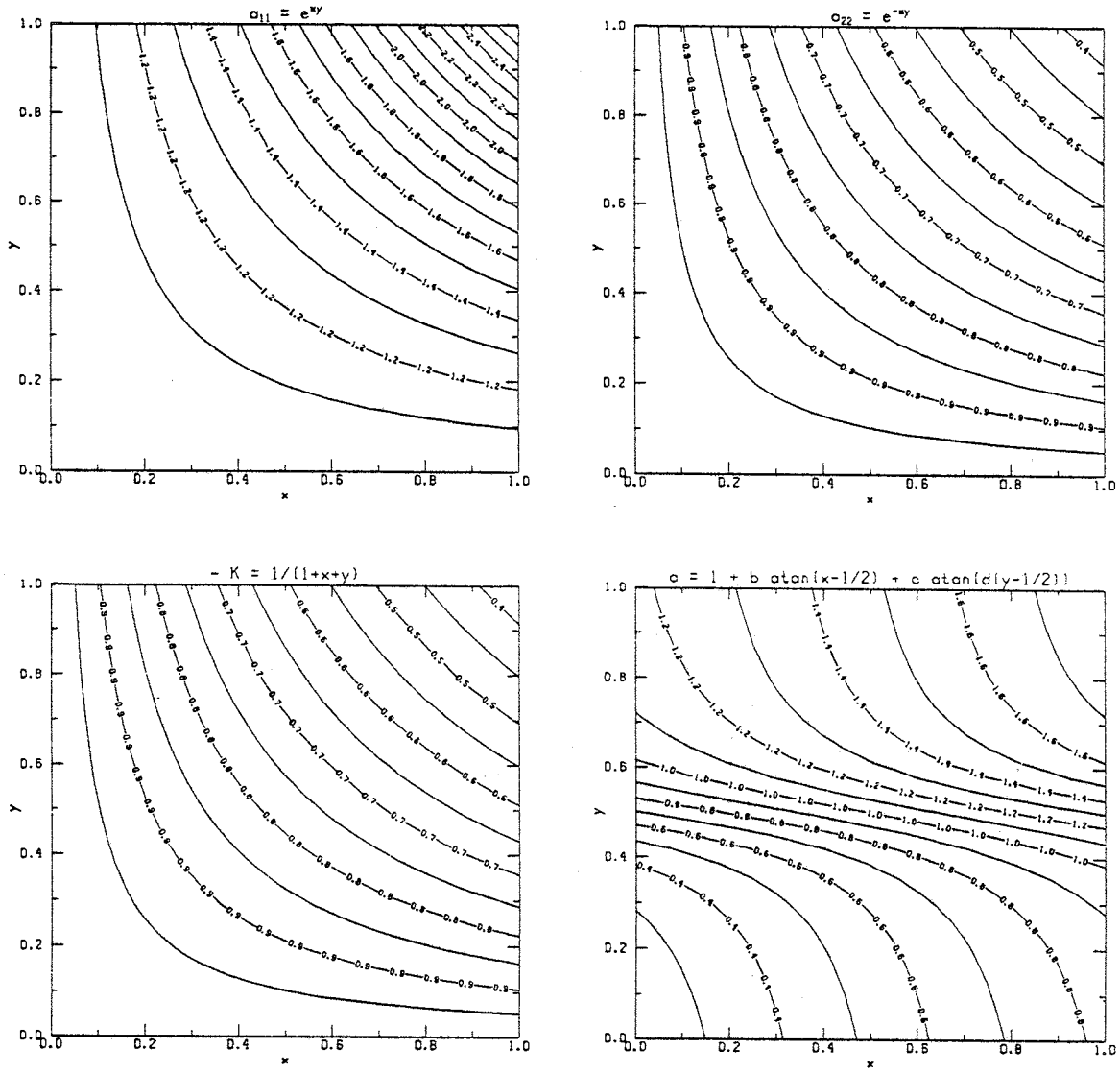


Figure 7: Contour plots of the coefficients of the fifth and sixth test problems, with nonseparable operators. (a) a_{11}, (b) a_{22}, (c) $-K$, (d) a.

h^{-1}		M_C	M_G	M_D	$M_{S(0)}$	$M_{S(1)}$	$M_{S(2)}$	GMM
8	I	7	7	7	7	7	7	7
	κ	2.419	2.422	2.528	2.412	2.411	2.412	2.419
	ρ	2.2(-1)	2.2(-1)	2.4(-1)	2.3(-1)	2.2(-1)	2.2(-1)	2.2(-1)
16	I	8	8	8	8	8	8	8
	κ	3.246	3.235	3.234	3.148	3.240	3.244	3.246
	ρ	3.0(-1)	3.1(-1)	3.1(-1)	3.2(-1)	3.1(-1)	3.1(-1)	3.0(-1)
32	I	9	9	10	10	10	10	9
	κ	3.917	3.892	4.041	4.565	3.992	4.034	3.917
	ρ	3.6(-1)	3.6(-1)	3.6(-1)	4.0(-1)	3.8(-1)	3.7(-1)	3.6(-1)

Table 5: $\nabla \cdot (\underline{a}\nabla u) + ku = f$ on the unit square, divided symmetrically into two strips, where $a_{11} = e^{xy}$, $a_{22} = e^{-xy}$, $a_{12} = a_{21} = 0$, and $K = (1 + x + y)^{-1}$, with subdomain preconditioning by the Laplacian. Results for PMM as a function of problem size and interface preconditioner and for GMM as a function of problem size.

to $\kappa(\tilde{B}^{-1}A)$, where $\tilde{B}$ is constructed according to (2.7), by using the different M-blocks shown at the head of the table columns. Results for the preconditioning arising from taking $\tilde{A}$ of (2.7) to be the Laplacian operator are shown in Table 5. This is a rather coarse-grained preconditioner, which does not take advantage of the sensitivity of the domain decomposition technique to local operator properties. We also consider a more conventional use of the PCG technique, the GMM of preconditioning a nonseparable problem by one which can be implemented by a fast Poisson solver over the entire domain. The results, displayed in the last column, are identical to those of M_C, which is, of course, the decomposed equivalent of the Laplacian.

The problem is a difficult one for all of the methods, as evidenced by the fact that for the coarsest mesh case, the number iterations for all methods is fully equal to the number of unknowns on the interface. (Of course, we are now iterating on vectors with length $O(h^{-2})$, but recall that for the first four problems considered, the PMM and SCM implementations have identical iteration counts much lower than the number of interfacial unknowns.) For problems of this difficulty, the MSC preconditioners do not fare appreciably worse than the optimal ones, and achieve comparable condition numbers even at the moderate resolutions investigated.

Table 6 shows the improvement possible by generating the preconditioner blocks $\tilde{A}_k$ by subdomain averaging the original operator A, giving a PMM finer granularity than a GMM. Let $\langle a_{11}\rangle_y(x)$ and $\langle K\rangle_y(x)$ denote subdomain averages of the respective coefficients with respect to y and $\langle a_{22}\rangle_x(y)$ and $\langle K\rangle_x(y)$ the same with respect to x. In each subdomain k, $\tilde{A}_k$ has the separable form

$$[\tilde{A}_k]_{ij} = \int_{\Omega_k} (\langle a_{11}\rangle_y \frac{\partial\psi_j}{\partial x}\frac{\partial\psi_i}{\partial x} + \frac{1}{2}\langle K\rangle_y \psi_j\psi_i + \langle a_{22}\rangle_x \frac{\partial\psi_j}{\partial y}\frac{\partial\psi_i}{\partial y} + \frac{1}{2}\langle K\rangle_x \psi_j\psi_i)\ \mathrm{dx}.$$

With the local averaging, the best of the optimal PMMs begin to break away from the GMM even though only two subdomains are employed. A more graphic illustration of the improvements offered by the PMM in the many subdomain case is given in example 3 of [4].

Our final stripwise decomposition example is

$$\nabla \cdot (a\nabla u) = f$$

on the unit square, where $a = 1 + b\tan^{-1}(x - \frac{1}{2}) + c\tan^{-1}(d(y - \frac{1}{2}))$. A contour plot of a with $b = 0.65$, $c = 0.35$, and $d = 10.0$ appears in Figure 7, and surface plots of the actual Schur complement matrix and various MSC approximations thereto were given in Figure 5.

h^{-1}		M_C	M_G	M_D	$M_{S(0)}$	$M_{S(1)}$	$M_{S(2)}$	GMM
8	I	5	5	6	5	5	4	5
	κ	1.465	1.465	1.975	1.456	1.373	1.310	1.524
	ρ	1.1(-1)	1.2(-1)	1.8(-1)	1.2(-1)	9.9(-2)	8.6(-2)	1.2(-1)
16	I	5	5	6	6	6	5	5
	κ	1.517	1.515	2.088	1.898	1.708	1.543	1.661
	ρ	1.3(-1)	1.3(-1)	1.9(-1)	2.0(-1)	1.5(-1)	1.3(-1)	1.5(-1)
32	I	5	5	6	8	7	7	6
	κ	1.515	1.513	2.143	2.637	2.325	2.060	1.812
	ρ	1.4(-1)	1.4(-1)	1.9(-1)	2.9(-1)	2.6(-1)	2.0(-1)	1.8(-1)

Table 6: $\nabla \cdot (\underline{a}\nabla u) + ku = f$ on the unit square, divided symmetrically into two strips, where $a_{11} = e^{xy}$, $a_{22} = e^{-xy}$, $a_{12} = a_{21} = 0$, and $K = (1+x+y)^{-1}$, with subdomain preconditioning by a locally averaged separable variable-coefficient operator. Results for PMM as a function of problem size and interface preconditioner and for GMM as a function of problem size.

h^{-1}		M_C	M_G	M_D	$M_{S(0)}$	$M_{S(1)}$	$M_{S(2)}$	GMM
8	I	6	6	7	6	5	5	7
	κ	1.871	1.891	2.408	2.003	1.786	1.813	2.794
	ρ	1.7(-1)	1.7(-1)	2.2(-1)	1.7(-1)	1.5(-2)	1.5(-2)	2.5(-1)
16	I	6	6	7	7	6	6	8
	κ	1.943	1.965	2.586	2.691	2.311	2.090	3.472
	ρ	2.0(-1)	2.0(-1)	2.5(-1)	2.6(-1)	2.1(-1)	1.8(-1)	2.9(-1)
32	I	7	7	7	9	8	7	9
	κ	2.096	2.121	2.618	3.795	3.247	2.777	3.852
	ρ	2.2(-1)	2.2(-1)	2.6(-1)	3.6(-1)	3.1(-1)	2.7(-1)	3.3(-1)

Table 7: $\nabla \cdot (a\nabla u) = f$ on the unit square, divided symmetrically into two strips, where $a = 1 + b\tan^{-1}(x - \frac{1}{2}) + c\tan^{-1}(d(y - \frac{1}{2}))$, with subdomain preconditioning by a locally averaged separable variable-coefficient operator. Results for PMM as a function of problem size and interface preconditioner and for GMM as a function of problem size.

A separable subdomain-averaged PMM preconditioner was constructed in the manner described above. In this example, the results of which appear in Table 7, all of the PMMs are better than or comparable to the GMM.

3.3. Boxwise decomposition tests

We carried out serial tests of the two decomposition strategies involving crosspoints on the first problem of the previous subsection, namely Poisson's equation on the unit square. Table 8 contains the results for the uncoupled crosspoint version of the algorithm, and Table 9 contains the results for the coupled crosspoint version. Since there is only one crosspoint in this example, the two algorithms differ only in the assembling of the right-hand side of the crosspoint equation, as described in §§2.7.

Even though there is only a single crosspoint, the difference between employing local and nonlocal basis functions for the crosspoint degree of freedom is evident in comparing the tables.

h^{-1}		M_C	M_G	M_D	$M_{S(0)}$	$M_{S(1)}$	$M_{S(2)}$
16	I	5	5	6	9	7	8
	κ	8.601	8.593	8.891	12.82	12.46	12.07
	ρ	6.9(-2)	6.9(-2)	2.0(-1)	2.7(-1)	2.4(-1)	2.2(-1)
32	I	6	6	7	10	10	10
	κ	12.36	12.36	12.13	24.39	22.70	21.90
	ρ	1.1(-1)	1.1(-4)	2.2(-1)	4.4(-1)	3.3(-1)	4.4(-1)
64	I	6	6	7	15	13	13
	κ	15.11	15.10	15.85	45.35	40.40	36.99
	ρ	2.0(-1)	2.0(-1)	2.4(-1)	5.3(-1)	4.8(-1)	4.8(-1)

Table 8: $\nabla^2 u = f$ on the unit square, divided into four equal boxes. Results for PMM *without* vertex coupling, as a function of problem size and interface preconditioner.

h^{-1}		M_C	M_G	M_D	$M_{S(0)}$	$M_{S(1)}$	$M_{S(2)}$
16	I	5	5	6	8	7	7
	κ	6.379	6.384	5.772	10.41	9.960	9.520
	ρ	6.7(-2)	6.8(-2)	1.9(-1)	2.8(-1)	1.9(-1)	2.4(-1)
32	I	5	5	6	11	9	11
	κ	7.130	7.232	8.201	19.83	18.31	16.69
	ρ	1.2(-1)	1.2(-1)	2.1(-1)	3.9(-1)	3.2(-1)	3.8(-1)
64	I	6	7	7	15	13	12
	κ	13.35	13.35	12.71	36.50	32.39	29.47
	ρ	2.1(-1)	1.5(-1)	2.2(-1)	5.0(-1)	4.6(-1)	4.6(-1)

Table 9: $\nabla^2 u = f$ on the unit square, divided into four equal boxes. Results for PMM with vertex coupling after Bramble *et al*, as a function of problem size and interface preconditioner.

The condition numbers are uniformly smaller in the nonlocal case, though the overall number of iterations is not greatly affected. The growth of condition number for Dryja's interface preconditioner with crosspoint coupling (the third column of Table 9) follows the theoretical bound $c_1(1+\ln(n/\sqrt{p})^2)$ with a c_1 of approximately 1.

A word is in order concerning the construction of the MSC preconditioners in the case of multiple interfaces. We require that M be block-diagonal as with the other preconditioners, to preserve the independence of the interfacial solves. In order to avoid solving four Dirichlet problems on each subdomain for each degree k of approximation in MSC(k), we economize by varying the components of the vectors v_i of (2.9b) on all interfaces simultaneously, rather than treating the interfaces one-by-one. The algorithm described in §§2.4 is then applied block-by-block to generate the coefficients of the interfacial blocks of $M_{S(k)}$. The vectors v_i always have zeros corresponding to the crosspoint location and the crosspoint block is unaffected by the choice of interfacial blocks. As in the case of Poisson's equation on two strips, the MSC methods cannot compete with the optimal methods.

3.4. Variational methods tests

The methods of §§2.5 were applied to the problem of Table 1. A two-strip decomposition was used for the saddle-point methods. The decomposition pictured in Figure 6, with an overlap region which spanned one subinterval on either side of the horizontal bisector of the square, was employed

h^{-1}		Saddle-1	Saddle-2	CGS/I	CGS/$I+\frac{1}{2}K$	CGS/$K^{1/2}$
8	I	3	3	4	3	3
	κ	2.750	2.325	8.411	3.096	1.673
	ρ	2.5(-2)	4.4(-2)	2.0(-4)	2.9(-3)	2.5(-3)
16	I	4	5	6	4	3
	κ	5.473	4.407	24.56	8.399	2.419
	ρ	8.4(-2)	1.1(-1)	1.9(-1)	4.6(-2)	1.1(-2)
32	I	7	7	11	6	3
	κ	10.63	8.652	81.65	27.39	4.011
	ρ	2.1(-1)	2.3(-1)	3.7(-1)	2.1(-1)	4.6(-2)

Table 10: $\nabla^2 u = f$ on the unit square, divided symmetrically into two strips, with and without overlap. Results as a function of problem size and method/preconditioning combination.

for the CG-Schwarz methods. (Thus, the width of the overlap region was allowed to shrink with increasing resolution.) The results are given in Table 10.

We note that the $K^{1/2}$ preconditioning is effective on the CG-Schwarz method, the spectrum of the discrete operator $\mathcal{A}$ of which has not been analyzed to date. However, it is apparent by comparison with Table 1 that on such simple problems these methods are inferior to SCM with optimal preconditioning.

4. Parallel experiments with domain decomposition algorithms

In order to test the convergence of the methods on a larger number of subdomains and to test the degree of parallelization that is possible in practice, two PMM methods, with and without vertex coupling and with M_D as the interfacial preconditioner, were programmed for the Intel Hypercube. For simplicity and economy of coefficient storage, we considered only Poisson's equation on the unit square. For programming convenience, efficient use of the processors and economy of data transfer between nodes, we wrote a single program to run on each of the nodes, rather than writing special purpose programs to do the major identifiable operations in Algorithm PCG, such as interior solves or edge solves. The same program was used to run both strips and boxes, with different compile-time constants to reshape the arrays. The number of processors was left dynamic.

4.1. Domain-to-processor mapping and dataflow analysis

The square domain is uniformly discretized with n subintervals on a side and divided into p subdomains in one of two ways. In the case of domains consisting of strips, each domain is an $n \times n/p$ rectangle. In the case of boxes, each domain is an $n/\sqrt{p} \times n/\sqrt{p}$ square. For simplicity and efficiency we require n and p to be powers of 2. By restricting ourselves to this limiting case of decomposition regularity, we are able to pass over the complex issue of load balancing. The work per processor is balanced *a priori* and does not change during the course of the solution process in response to any adaptive mechanism. It should be noted that regular decompositions of this type are the exception in applications. The tradeoffs between load balancing efficiency and communication efficiency in multiprocessor implementations of adaptive algorithms for partial differential equations have begun to receive attention elsewhere ([3, 18]). We note however, that some geometrical adaptivity can be accommodated within the context of *logically* uniform decompositions by means of generalized tensor-product gridding (see, *e.g.*, the examples in [4]).

Four types of nodes are to be distinguished in the decomposition, for purposes of keeping track of the data flow: interior nodes, separator set nodes, nodes which are adjacent to a separator, and nodes which are adjacent to an exterior boundary. For the strip decomposition, we associate the

gridpoints along each interface with one of the two subdomains it separates so that each processor (except for one) is assigned one of the interfaces of length n. For definiteness in what follows, let the strips run north-south, and let the east face of each subdomain contain the separator nodes, the west face containing nodes which are adjacent to a separator, with the obvious exceptions of the west-most and east-most subdomains. For the box decomposition, we associate the gridpoints along each interface with one of the two subdomains it separates and each vertex with one of the four subdomains it separates so that each processor (except for those along two of the domain boundaries) is assigned two interfaces of length $n\sqrt{p}$ and one crosspoint. For definiteness, let the east face and south faces of each subdomain contain the separator nodes, let the southeast corner contain the crosspoint, and let the west and north faces contain nodes which are adjacent to a separator, again with the obvious boundary exceptions. We map p strips onto the hypercube by labeling the strips from one end of the domain to the other in binary reflected Gray code order [19]. We map p boxes onto the hypercube by forming the tensor-product of two such rings, along the row and column directions of the subdomain "grid". In this way all of the local exchanges in the physical domain are also local exchanges between processors, any two of which are connected if and only if their binary representations differ in exactly one bit.

We now analyze the data flow in each of the major steps in Algorithm PCG. We concentrate on the boxwise decomposition, since strips may be regarded as a special case thereof.

The operation (*PCG*.7) of applying the distributed $\mathcal{A}$ matrix consists of: (1) sending the vector of data along each processor boundary not adjacent to a physical boundary to the neighboring processor; (2) computing the interior components of the product from the 5-point star formula; (3) receiving the vector of data coming across each boundary and computing the boundary components of the product. Note that it is not necessary to distinguish between separator nodes and those adjacent to a separator in this step.

The operation (*PCG*.11) of solving with $\mathcal{B}$ consists of: (1) solving for u^P in the subdomain interior (the subdomain interior includes the processor boundary nodes that are adjacent to a separator); (2) sending the vector of data along each north and west processor boundary not adjacent to a physical boundary to the neighboring processor; (3) receiving the vector of data coming across each south and east processor boundary and forming the right-hand side of the edge separator system and the local summands of the right-hand side of the crosspoint system; (4) sending the appropriate scalar summands to the processors to the north and west, if any; (5) solving the south and east edge systems for u^E on the separator set; (6) receiving the summands coming from the processors to the south and east, if any, and forming the right-hand side of the crosspoint system; (7) scattering the local component of the right-hand side of the crosspoint system to all other processors, and receiving their components in turn; (8) solving an identical global crosspoint system in each processor; (9) extending the crosspoint data linearly along the separator set edges to form u^V; (10) summing u^V and u^E along the separator set to form u^H on the separator set; (11) sending the vector of u^H data along each south and east boundary not adjacent to a physical boundary to the neighboring processor; (12) receiving the vector of data coming across each north and east processor boundary and forming the right-hand side of the interior u^H system; (13) solving for u^H in the subdomain interior and summing u^H and u^P. Steps (4), (6)-(7), and (9)-(10) are unnecessary in the decoupled form of the algorithm, and step (8) is replaced by a scalar divide local to each processor containing a crosspoint in this case.

The operation of performing a dot product in (*PCG*.7) and (*PCG*.12) consists of two traversals of a binary spanning tree, arbitrarily rooted in processor "0". First, the local portion of the dot product is computed. These partial sums are then passed from leaf to root, and summed along the way. The root processor performs the final sum and sends the data back from root to leaf.

h^{-1}		$p=1$	2	4	8	16
8	I	1				
	κ	1.000				
	ρ	2.5(-6)				
	$\|\|r^I\|\|$	8.1(-6)				
	T	1.408				
	s	-				
16	I	1	3			
	κ	1.000	1.260			
	ρ	5.0(-6)	2.9(-2)			
	$\|\|r^I\|\|$	1.7(-5)	8.3(-5)			
	T	7.680	4.288			
	s	-	*1.79*			
32	I	1	2	5		
	κ	1.000	1.092	3.292		
	ρ	1.7(-5)	8.8(-3)	9.9(-2)		
	$\|\|r^I\|\|$	3.5(-5)	3.0(-4)	3.2(-5)		
	T	39.25	16.83	10.66		
	s	-	*2.33*	*1.58*		
64	I		2	4	8	
	κ		1.091	3.291	12.98	
	ρ		4.8(-3)	8.5(-2)	2.8(-1)	
	$\|\|r^I\|\|$		7.7(-5)	1.1(-4)	4.8(-5)	
	T		86.82	47.71	29.20	
	s		-	*1.82*	*1.64*	
128	I				8	18
	κ				12.98	51.88
	ρ				2.8(-1)	5.9(-1)
	$\|\|r^I\|\|$				2.7(-5)	3.3(-5)
	T				157.1	115.7
	s				-	*1.96*

Table 11: $\nabla^2 u = f$ in the unit square, divided into equal strips. Results for PMM with M_D on the interfaces, as a function of problem size and number of processors.

Our present implementation of the global scatter required in distributing the right-hand side of the crosspoint system, in step (7) of (*PCG*.11), consists of a different scatter rooted in each node which is identical to the second half of the dot product routine described above.

4.2. Numerical results

We present data obtained on an Intel iPSC-7 for the strip form and the two box forms of the algorithm described above. For simplicity of coefficient generation and economy of storage (as memory proved to be constraining), we considered only Laplace's equation on a square. The problem is the same as that in Tables 1, 8, and 9.

Our conventions for the tests on the iPSC differ from those in §3 in that the preconditioned residual is used to monitor convergence, rather than the actual residual. This is the practical choice since we are not comparing different preconditioners against each other in this section, but rather carrying out larger scale tests on selected algorithms. Since monitoring the true residual would

require an extra global dot product per iteration, the reported execution times would exhibit an unnatural penalty associated with increased subdivision of the domain. However, for completeness, we do report the unpreconditioned residual at termination.

The notation employed in the tables follows that of section 3 except that ρ is now defined as $((r^I, \mathcal{B}^{-1}r^I)/(r^0, \mathcal{B}^{-1}r^0))^{1/(2I)}$. T is the total execution time (computation plus communication), including the parallelized computation of the discrete forcing term and other initialization overhead. s, where applicable, measures the ratio of two execution times: T for the next smaller number of processors on the same size problem divided by the current T. It is thus a local measure of speedup, and a natural one for domain decomposition algorithms, but it is not the *speedup* as it is defined in a pure sense (see §5).

We comment briefly on the sparsity of the tables. Our program requires a minimum of 8 subintervals on a side for each subdomain. Therefore, the number of processors that can be employed on a given problem is bounded above by the resolution of the mesh. Unfortunately, the number of processors is also bounded below by the limited local memory of each node. The largest number of degrees of freedom that could be accommodated per subdomain was between 2^{11} and 2^{12}; therefore, our largest problems, involving 2^{14} or 2^{16} nodes, could not be run on small numbers of processors. These two restrictions imposed a banded structure on the region of $h-p$ parameter space in which it was feasible to run experiments.

The results of a series of runs using strips are given in Table 11. The $p = 2$ column of this table is comparable to the M_D column of Table 1; however, the more lenient convergence criterion leads to one less iteration in the more highly refined cases, and the Lanczos estimate of the condition number is obviously sensitive to I for I as small as 2 or 3 in this problem. Note the effect of Theorem 2.2: as the number of processors is increased while the refinement is held constant, κ goes up asymptotically in proportion to the square of the number of processors (subdomains). The number of iterations required does not behave as badly as the condition number. From the limited data, it appears to go up roughly in proportion to the number of processors, with fixed refinement. The increasing number of iterations asymptotically defeats the power of the additional processors as far as s is concerned, however.

The results of a series of runs using boxes without crosspoint coupling are given in Table 12. The $p = 4$ column of this table is may be compared to the M_D column of Table 8; again the preconditioned residual convergence criterion is more lenient, but the condition number estimates agree almost exactly. In this table d/h is constant along a diagonal, therefore by Theorem 2.3 we would expect to see a proportionality between κ and p along a diagonal, which is approximately the case. The number of iterations required goes up sublinearly with the number of processors (subdomains), with fixed refinement.

The results of a series of runs using boxes with crosspoint coupling are given in Table 13. The $p = 4$ column of this table may be compared to the M_D column of Table 9, subject to the caveats above, and the $p = 16$ column shows close agreement with Table 6.2 of [4]. According to Theorem 2.3, the condition number should be bounded by a constant along a diagonal, which appears to be the case. The number of iterations virtually levels off and the condition number actually *decreases* as more processors are added with fixed refinement.

5. Domain decomposition on parallel computers

This section emphasizes the application of domain decomposition to parallel computation. Thus, we consider an abstract problem with no special features such as complicated domain geometry or operator variability to dictate the decomposition and ask what strategy leads to the most efficient parallel implementation. Several interesting questions remain, of which the most important are: how much of a speedup over a single processor can be obtained in practice, and how is the performance of the parallel algorithm affected by decomposition topology, architecture topology,

h^{-1}		$p=1$	4	16	64
8	I	1			
	κ	1.000			
	ρ	2.5(-6)			
	$\|\|r^I\|\|$	8.1(-6)			
	T	1.408			
	s	-			
16	I	1	6		
	κ	1.000	8.891		
	ρ	5.0(-6)	1.6(-1)		
	$\|\|r^I\|\|$	1.7(-5)	4.6(-5)		
	T	7.680	4.736		
	s	-	*1.62*		
32	I	1	6	11	
	κ	1.000	12.13	25.27	
	ρ	1.7(-5)	1.8(-1)	3.6(-1)	
	$\|\|r^I\|\|$	3.5(-5)	1.1(-4)	2.3(-5)	
	T	39.25	21.20	9.184	
	s	-	*1.85*	*2.81*	
64	I		6	12	17
	κ		15.85	32.18	94.39
	ρ		1.8(-1)	4.0(-1)	5.7(-1)
	$\|\|r^I\|\|$		8.4(-5)	2.7(-5)	5.9(-4)
	T		104.3	38.70	15.74
	s		-	*2.70*	*2.46*
128	I			13	19
	κ			40.53	119.7
	ρ			4.7(-1)	6.1(-1)
	$\|\|r^I\|\|$			7.0(-5)	6.8(-5)
	T			196.6	60.91
	s			-	*3.23*
256	I				22
	κ				145.6
	ρ				6.5(-1)
	$\|\|r^I\|\|$				4.4(-5)
	T				320.4
	s				-

Table 12: $\nabla^2 u = f$ in the unit square, divided into equal boxes. Results for PMM with M_D on the interfaces and *without* vertex coupling, as a function of problem size and number of processors.

machine parameters, and algorithmic options. To begin to answer these questions, we construct theoretical computational complexity models of various approaches and compare them. One of our purposes is to determine what a parallel computer suitable for domain decomposition algorithms should look like. This includes not only the architecture, but the relative speeds of communication and floating point.

h^{-1}		$p=1$	4	16	64
8	I	1			
	κ	1.000			
	ρ	2.5(-6)			
	$\|\|r^I\|\|$	8.1(-6)			
	T	1.424			
	s	-			
16	I	1	6		
	κ	1.000	5.739		
	ρ	5.0(-6)	1.3(-1)		
	$\|\|r^I\|\|$	1.7(-5)	2.3(-5)		
	T	7.680	4.816		
	s	-	*1.60*		
32	I	1	5	7	
	κ	1.000	7.879	7.005	
	ρ	1.7(-5)	1.5(-1)	2.6(-1)	
	$\|\|r^I\|\|$	3.5(-5)	3.0(-4)	2.7(-4)	
	T	39.26	18.27	8.016	
	s	-	*2.15*	*2.28*	
64	I		6	7	6
	κ		10.72	10.24	7.163
	ρ		1.5(-1)	2.5(-1)	2.1(-1)
	$\|\|r^I\|\|$		4.8(-5)	5.2(-4)	5.5(-4)
	T		104.5	25.84	19.09
	s		-	*4.04*	*1.35*
128	I			7	7
	κ			14.10	10.53
	ρ			2.5(-1)	2.4(-1)
	$\|\|r^I\|\|$			5.3(-4)	3.3(-4)
	T			114.5	39.12
	s			-	*2.93*
256	I				8
	κ				14.50
	ρ				2.7(-1)
	$\|\|r^I\|\|$				2.2(-4)
	T				141.9
	s				-

Table 13: $\nabla^2 u = f$ in the unit square, divided into equal boxes. Results for PMM with M_D on the interfaces and with vertex coupling after Bramble *et al*, as a function of problem size and number of processors.

In any iterative method two distinct factors contribute to the cost: the cost of a single iteration and the number of iterations required. In this section, we will look only at the cost per iteration for a given algorithm, since this is the only factor that depends on the parallelizability of the algorithms. Of course, the iteration count is crucial in deciding which algorithm is most suitable for a given problem, and this area been explored theoretically and experimentally in the preceding sections.

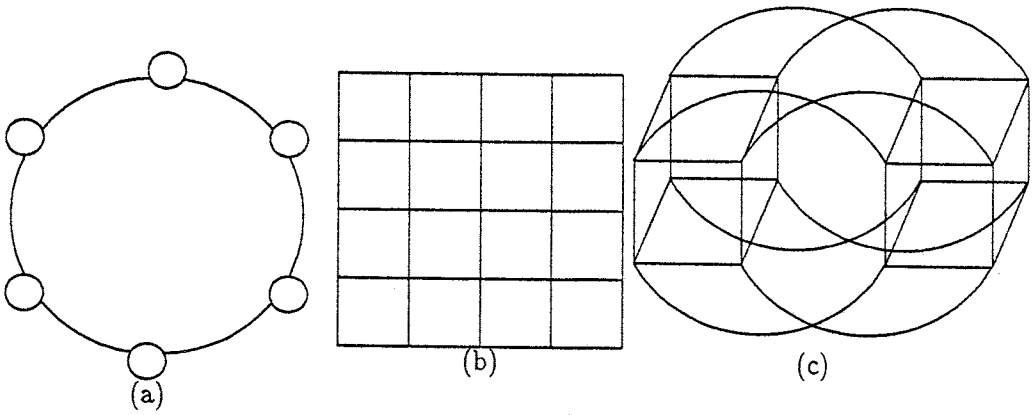


Figure 8: The three parallel architectures under consideration. (a) is a ring; the processors are indicated by circles. (b) is a mesh; the processors are at the intersections of the lines. (c) is a 4-D hypercube; again the processors are at the intersections.

We continue to consider two basic types of decomposition topologies: strips and boxes. We consider one-to-one subdomain-to-processor maps and estimate the costs for a single iteration. For comparison, we consider the same algorithm implemented on a single processor with the same number of subdomains. This allows us to see only how effectively domain decomposition algorithms can be parallelized. Whether domain decomposition is the best possible parallel algorithm for elliptic systems is beyond the scope of this paper.

As described in §§4.1, we consider a square domain uniformly discretized with n subintervals on a side and we assume a total of p processors which evenly divide the domain into either strips or boxes. We further assume that the $\mathcal{A}$ matrix comes from a 5-point stencil, that the $\mathcal{B}$ matrix solves can be done with a fast Poisson solver, and that the edge solves can be done with FFT's.

5.1. Parallel processor model and subdomain-to-processor mapping

In any parallel computer, interprocessor communication must be considered. We will assume that the time to transmit a message of length n to a "neighbor" takes time $\alpha + \beta n$. A "neighbor" is a processor which is directly connected. We will further assume that only sending messages takes time; receiving is free. α is the *latency* and β is the *transfer time*. A floating point operation takes time f.

We consider three different parallel architectures: a ring (cf. Figure 8(a)), a two dimensional mesh with wrap-around (cf. Figure 8(b), wrap-around not shown), and a hypercube (cf. Figure 8(c)), which embeds the first two. The mappings of the strips onto the ring or the hypercube and the boxes onto the mesh or the hypercube are the natural ones discussed in §§4.1. Because of the local connectivity of the discrete elliptic operator, the only real difference between these architectures for a given decomposition type is their performance on the global dot products in the Conjugate Gradient algorithm in steps (*PCG*.7) and (*PCG*.12). The purpose of our estimates is to suggest optimal relative sizes of α, β, and f for efficient use of the processors.

5.2. Complexity estimates

Using the model described above, we estimate the complexity of a computation using a decomposition into either strips or boxes on the three architectures under consideration. The times are shown in Table 14 on a *per* subdomain *per* iteration basis. All of the times shown in the "Comm. Cost" column represent transmissions of edge data from one processor to a neighbor. Note that since we assume that nearest neighbor communications are done one at a time, there is no difference between the hypercube, the mesh, and the ring for sends to a neighbor (at most, there is a constant

Decomposition	Operation type	Comm. Cost	Comp. Cost	Quantity
Single Processor (p domains) Strips	Dot Product	0	n^2	2
	$\mathcal{A}$-multiply	0	$5n^2$	1
	$\mathcal{B}$-solve			
	interior	0	$C_I n^2 p^{-1} \log(np^{-1})$	2
	edge	0	$C_E n \log n$	1
Single Processor (p domains) Boxes	Dot Product	0	n^2	2
	$\mathcal{A}$-multiply	0	$5n^2$	1
	$\mathcal{B}$-solve			
	interior	0	$C_I n^2 p^{-1} \log(np^{-\frac{1}{2}})$	2
	edge	0	$C_E n p^{-\frac{1}{2}} \log(np^{-\frac{1}{2}})$	2
	corner	0	1	1
Parallel Processor Strips	Dot Product	see Table 15	$p + n^2 p^{-1}$	2
	$\mathcal{A}$-multiply	$2(\alpha^* + \beta^* n)$	$5n^2 p^{-1}$	1
	$\mathcal{B}$-solve:			
	interior	0	$C_I n^2 p^{-1} \log(np^{-1})$	1
	edge	$\alpha^* + \beta^* n$	$C_E n \log n$	1
	interior	$\alpha^* + \beta^* n$	$C_I n^2 p^{-1} \log(np^{-1})$	1
Parallel Processor Boxes	Dot Product	see Table 15	$p + n^2 p^{-1}$	2
	$\mathcal{A}$-multiply	$4(\alpha^* + \beta^* n p^{-\frac{1}{2}})$	$5n^2 p^{-1}$	1
	$\mathcal{B}$-solve:			
	interior	0	$C_I n^2 p^{-1} \log(np^{-\frac{1}{2}})$	1
	edge	$(\alpha^* + \beta^* n p^{-\frac{1}{2}})$	$C_E n p^{-\frac{1}{2}} \log(np^{-\frac{1}{2}})$	2
	corner	0	1	1
	interior	$2(\alpha^* + \beta^* n p^{-\frac{1}{2}})$	$C_I n^2 p^{-1} \log(np^{-\frac{1}{2}})$	1

Table 14: Times (in units of f) for communication and computation for the partitioned matrix method on two decomposition topologies and various forms of parallel architectures ($\alpha^* = \alpha/f$, $\beta^* = \beta/f$). The only differences between processors are in the times for dot-products, which are shown in Table 15. Also included are times for a single processor implementation of the domain decomposition algorithm.

Architecture	Dot product time
Ring	$p(\alpha + \beta)$
2-D Mesh	$2\sqrt{p}(\alpha + \beta)$
Hypercube	$2 \log p(\alpha + \beta)$

Table 15: Communications times for doing dot products. These assume that a single node collects the data, performs the dot product, and distributes the result.

factor of four difference here between the architectures if simultaneous sends are allowed). It is in the global dot product that the three architectures differ most; these times are shown in Table 15. All of these times are for simple dot-product algorithms, and we make no claim that they are optimal. However, doing much better is hard; for example, see [27] for a sophisticated method for doing

Architecture	Strips	boxes
Hypercube	$r = \frac{n - \log p}{\log p - 1}$	$r = \frac{2n/\sqrt{p} - \log p}{\log p - 2}$
Mesh	NA	$r = \frac{2n - p}{p - 2\sqrt{p}}$
Ring	$r = \frac{2n - p}{p - 2}$	NA

Table 16: Value of $\alpha/\beta = r$ where local and global communication costs are the same.

the global dot product which is only a little faster than the "obvious" algorithm we have used here. The times shown in the "Comp. Cost" column represent the leading order operation counts (in scalar multiply-adds) of the various "Operation types" only, and are accurate for asymptotically large n. When p is simultaneously large, lower-order terms in n should arguably be retained, but we have not made the attempt here. Thus the apparent reduction in serial complexity in going from a global-domain fast Poisson solves (cost $C_I n^2 \log n$) to a set of p subdomain fast Poisson solve (cost $p \cdot C_I n(np^{-1}) \log(np^{-1})$) should not be regarded as an advantage that can be exploited by taking p large. The $O(n^2)$ terms of the fast Poisson solver become important in this limit.

While it is tempting to discard all of the communication terms because they are asymptotically in n unimportant, it turns out that because n and p may be roughly the same size and that α and/or β may be much larger than f, this can be very misleading. In the rest of this section, we will look at where in the parameter space of n, p, α, β and f local or global communication is as expensive as computation. We will do this by first determining where local or global communication dominates the communication cost, then looking at where either local or global communication dominates computation. Finally, we will estimate the performance of several possible computer architectures by picking representative values of α, β, and f.

The local communication consists only of edge exchanges, and takes time $4(\alpha + \beta n)$ for strips and time $8(\alpha + \beta n/\sqrt{p})$ for boxes. The global communication consists only of dot products, and is of the form $C_D(\alpha + \beta)$, where $C_D = 2p$ for a ring, $C_D = 4\sqrt{p}$ for a mesh, and $C_D = 4 \log p$ for a hypercube.

We can determine the ratio $r = \alpha/\beta$ where these two costs are equal as a function of n and p. The formulas are shown in Table 16. Contour plots of the four cases presented in Table 16 are shown in Figure 9. These plots clearly show that a hypercube can afford a higher ratio of α/β than either a ring or a mesh, but that (particularly for boxes) the advantage is not very great for typical ranges of p and n.

Also of interest is where the ratios α/f and β/f are such that the communication time is equal to the computation time. We can look at this by considering just the dominant terms in the communication and the computation.

The dominant computation term for strips is $2C_I n^2 p^{-1} \log(np^{-1})$ and the dominant communication term is either the local term $4(\alpha + \beta n)$ or the global term $C_D(\alpha + \beta)$. The dominant computation term for boxes is $2C_I n^2 p^{-1} \log(np^{-\frac{1}{2}})$, and the dominant communication term is either the local term $8(\alpha + \beta np^{-\frac{1}{2}})$ or the global term $C_D(\alpha + \beta)$. Table 17 summarizes these results and Figure 10 gives contour plots of these values. To generate these and subsequent results, we have for convenience set $C_I = C_E = 5$, which are representative of cyclic reduction for separable symmetric discrete elliptic systems and a pair of sine transforms, respectively.

Because of the complexities of these formulas, we have constructed graphs of the speedup, fraction spent in communication, and efficiency for several hardware implementations (values of α, β, and f) for each of these architectures. We define the speedup as the ratio of the time to do a single

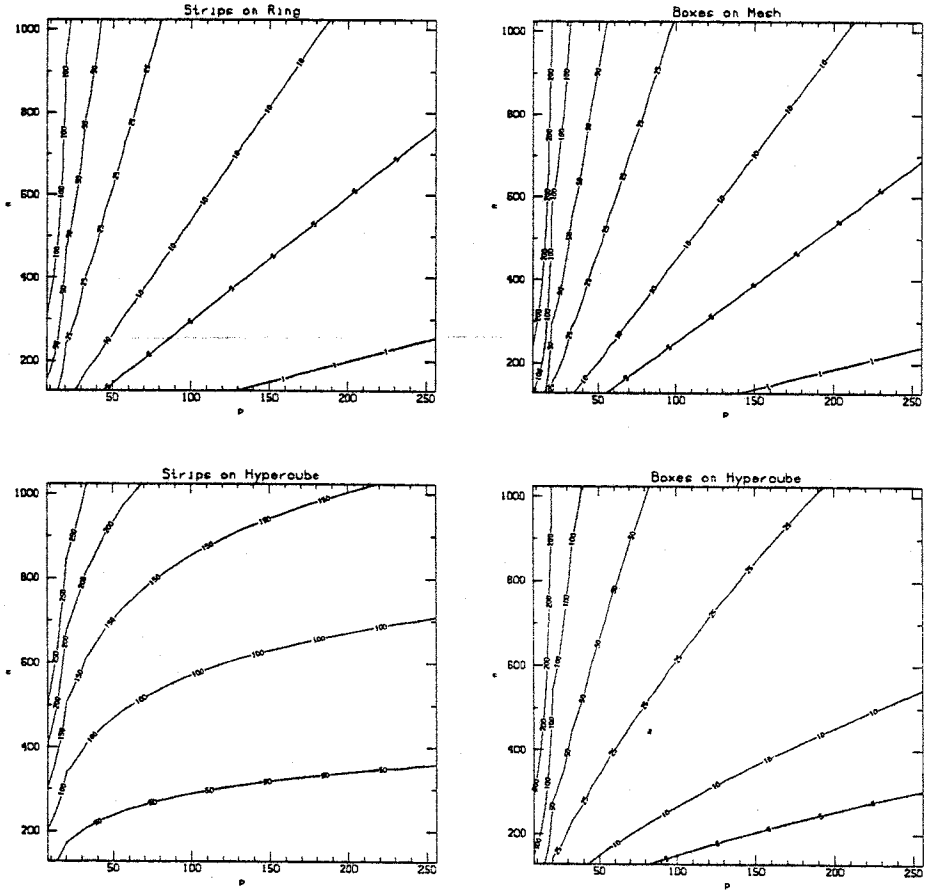


Figure 9: Contour plots of $r \equiv$ ratio of α/β where global communication takes the same time as local communication. If for a given (p, n), α/β is larger than the value on the plot, then global communication is more important; otherwise, local communication is dominant. Alternatively, for a given r, local communication dominates above the contour, and global communication below.

Architecture	Strips		Boxes	
	local	global	local	global
Hypercube	$\frac{f}{\beta} \approx \frac{2p}{5n\log(np^{-1})}$	$\frac{f}{\alpha} \approx \frac{2p\log p}{5n^2\log(np^{-1})}$	$\frac{f}{\beta} \approx \frac{4p^{\frac{1}{2}}}{5n\log(np^{-\frac{1}{2}})}$	$\frac{f}{\alpha} \approx \frac{2p\log p}{5n^2p^{-1}\log(np^{-\frac{1}{2}})}$
Ring	see Hypercube	$\frac{f}{\alpha} \approx \frac{p^2}{5n^2\log(np^{-1})}$	NA	NA
Mesh	NA	NA	see Hypercube	$\frac{f}{\alpha} \approx \frac{2p^{\frac{3}{2}}}{5n^2\log(np^{-\frac{1}{2}})}$

Table 17: Ratios of machine parameters where communication takes as much time as computation.

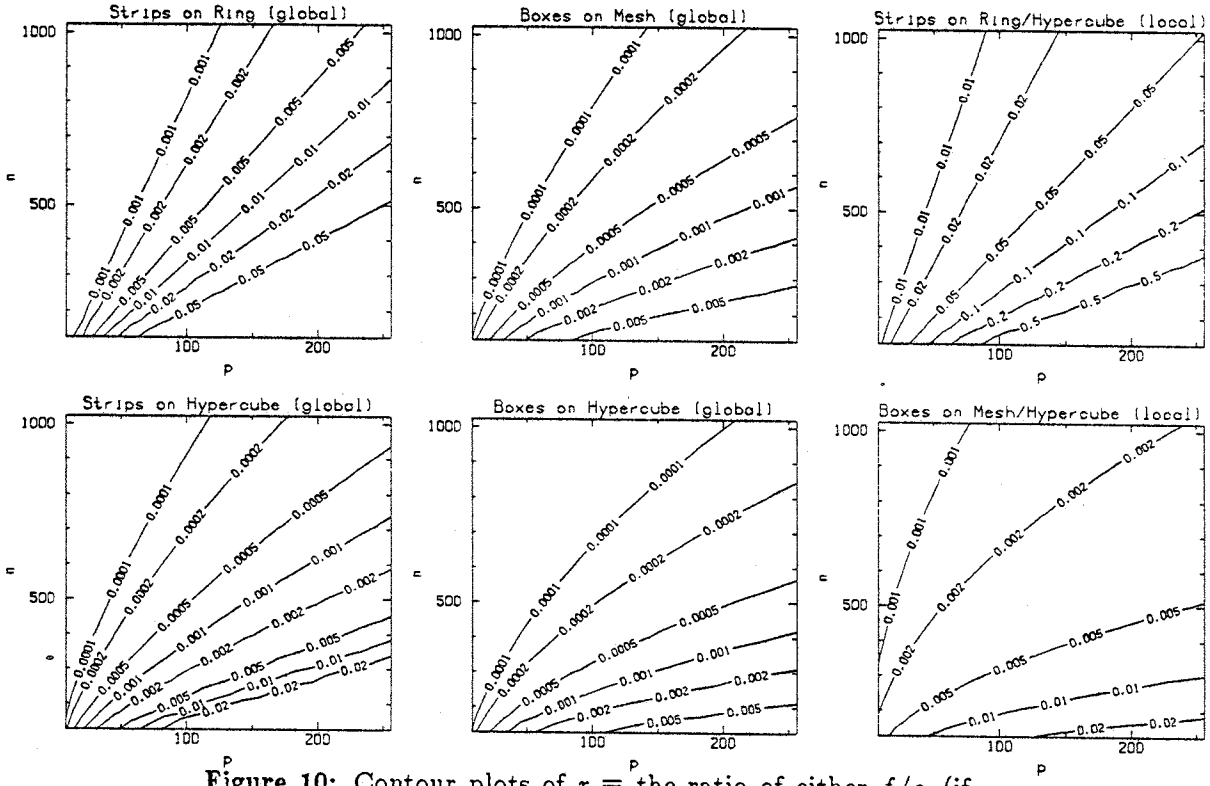


Figure 10: Contour plots of $r \equiv$ the ratio of either f/α (if global communication is dominant) or f/β (if local communication is dominant). If f/α is greater than the value in the contour plot then communication takes more time than computation when global communication is dominant. Alternatively, for a given r, computation dominates above the contour, and communication below. Similarly for local communication.

serial iteration to the time to do a single parallel iteration for the same decomposition. The fraction spent in communication is the ratio of the communication time over the total time (computation plus communication). The efficiency is the speedup divided by the number of processors; it measures

how well a parallel computer is being used and gives an indication of how much value there is in increasing the number of processors.

In order to provide some comparisons, we will consider all four combinations of parallel architecture and domain decomposition with the same parameters. This is somewhat unfair to the ring and to the mesh, since their simpler communication structure would allow more engineering effort to be spent on faster communication or computation. The values we have chosen are meant to be fairly representative of what exists or could exist in the near future.

These estimates of which part of the algorithm dominate performance suggest a few sample choices of the parameters α, β, and f. The most favorable is a machine which is not communication dominated. Such a processor might realistically be constructed to have:

$$\alpha = 100\mu \text{ seconds}$$
$$\beta = 5\mu \text{ seconds}$$
$$f = 10\mu \text{ seconds}$$

This makes each node a 100 Kflop processor, with 200 Kiloword/second transfer speed, and an I/O startup time of 10 floating point operations. This machine has a relatively well balanced communication and computation unit, and probably represents a good compromise. Of course, in all cases it is the dimensionless ratios of communication time to computation time, or α/f and β/f which matter. The speed of identical ensemble architectures with the same ratios will scale linearly with the basic floating point speed. For the balanced processor, these values are $\alpha/f = 10$ and $\beta/f = 0.5$. Figures 11–12 show various results based on the complexity estimates of Table 1 for this machine, as functions of n and p, where n ranges from 2^7 to 2^{10} and p from 1 to 2^8. The data is set to zero over the regions in which $p > n$ Note that, for these parameters, there is little difference between a ring or a mesh and a hypercube (cf. results in Tables 16 and 17). We emphasize that these results are all theoretical. Note that even with this balanced machine, communication is not unimportant if decomposition into strips is used, because of the long vectors of interface data which must be moved between processors. For boxes, whose area-to-perimeter ratio is better for large p, the communication cost rapidly vanishes for large problems (which can effectively put to use large numbers of processors). An effect which we do not take into account here is the potential for vectorization *within* a subdomain. Because of the longer vector lengths, stripwise decompositions have a tendency to be favorable from the computational standpoint if tasks local to a processor vectorize well.

The first two figures show the speedup in surface plot and contour plot form, the next the fraction of the time taken in communication, and the last shows the efficiency. A few remarks are in order about these graphs. Recall that all are on a *per iteration* basis.

A meaningful speedup is difficult to define. One measure would be to compute the ratio of the time for the *best* single processor algorithm to the time for a parallel domain decomposition algorithm. Unfortunately, there is no consensus for the best single processor algorithm. Instead, we consider the ratio of the time for a single processor implementation of the domain decomposition algorithm, with the same number of domains as the parallel implementation. This speedup measures the opposing effects of less work per processor versus more communication.

The plot of the fraction of the time in communication shows quite graphically the overhead due to the distribution of the algorithm across many processors.

The plot of the efficiency provides a complementary visualization of this distribution overhead, which is best amortized over large subdomains from the point of view of the per iteration cost factor.

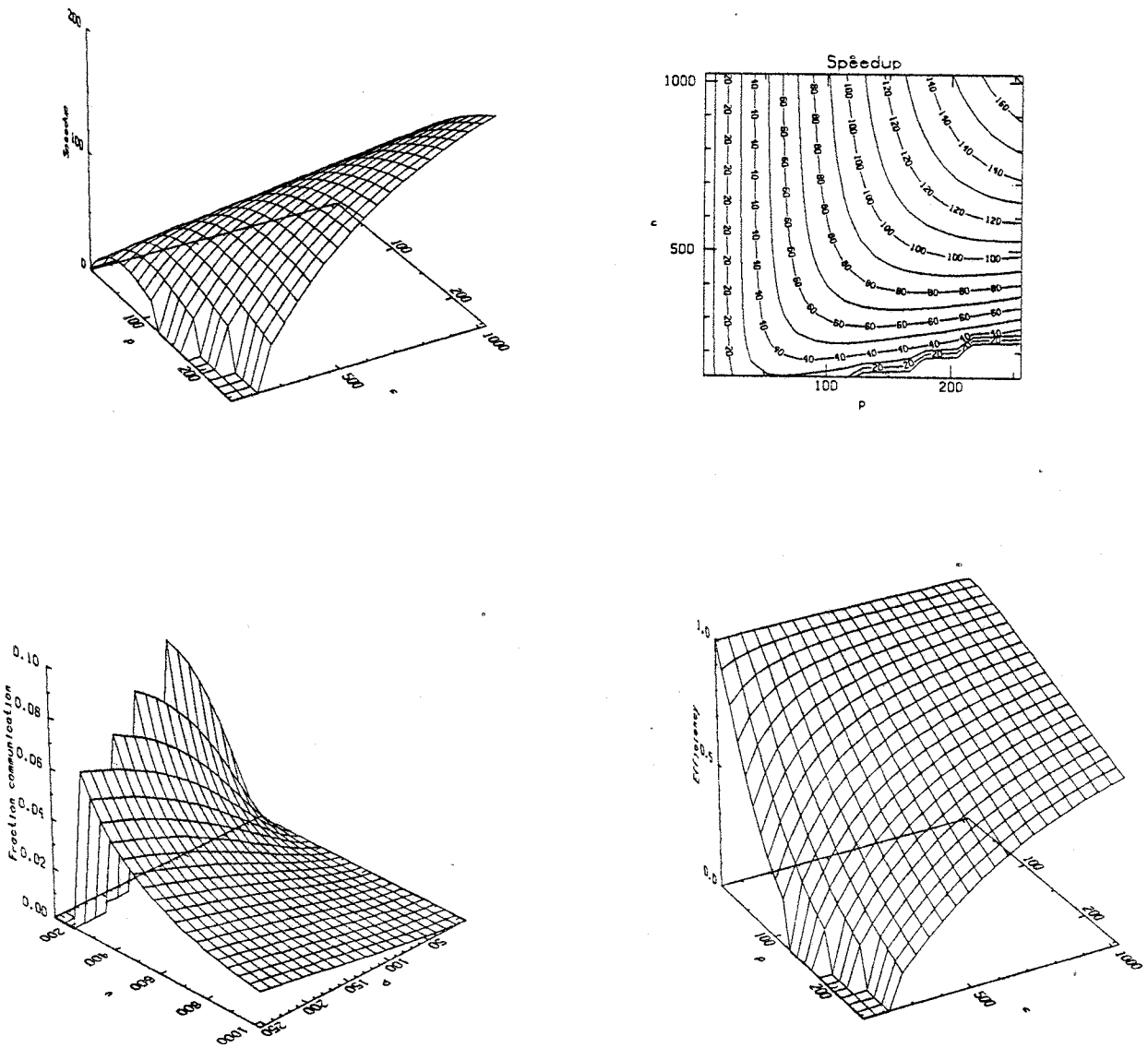


Figure 11: Theoretical parallelization measures for a hypercube, using strips for the domain decomposition. The figures for a ring using strips are virtually identical. Balanced processor.

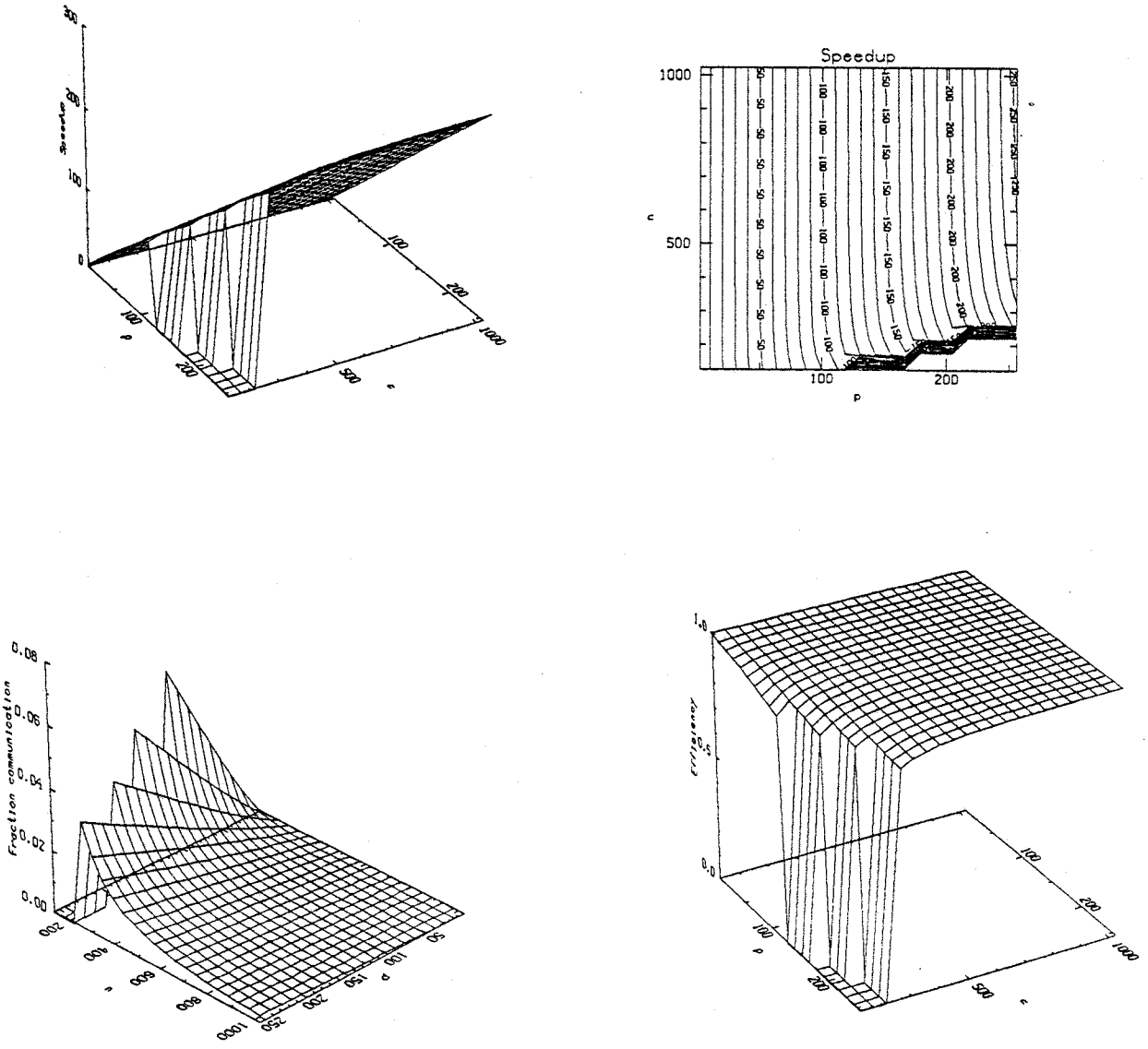


Figure 12: Theoretical parallelization measures for a hypercube, using boxes for the domain decomposition. The figures for a mesh using boxes are virtually identical. Balanced Processor.

Another choice for the parameters would represent a faster processor, purchased at the expense of communication startups. Such a processor might have parameters:

$$\alpha = 2500\mu \text{ seconds}$$
$$\beta = 2.5\mu \text{ seconds}$$
$$f = 5\mu \text{ seconds}$$

Thus, $\alpha/f = 500$, $\beta/f = 0.5$, and $\alpha/\beta = 1000$. Note that in such a machine, global communication will dominate local communication (cf. Figure 9) and it will often be communication dominated (cf. Figure 10), particularly for strips. The results for this processor are shown in Figures 13–16.

For simplicity, all of the plotted results for decompositions of box type are based on the communication-free form of the crosspoint solves which are needed as part of applying B^{-1}. As discussed theoretically in §2 and confirmed experimentally in §§4.2, this preconditioning allows the overall iteration count to grow as p increases. Therefore, we consider the more implicit algorithm due to Bramble *et al* [4]. This method requires that a linear system of size p with bandwidth $\sqrt{p}$ involving values at all the corner points be solved at each iteration. The computational cost of assembling and solving this extra system of equations is $O(n/\sqrt{p}) + O(p^2)$. This is negligible for practical ranges of p and n in comparison to the computational work required for the interior solves. However, the communication required to set up the right-hand side demands a close look. Each processor must first exchange a scalar with each of the two neighbors whose interfaces emanate from its crosspoint to form its component of the right-hand side. Then all of the data associated with each crosspoint must be scattered to *every* processor, each of which solves an identical system. Because of the small size of the system, this is faster than distributing the matrix across all the processors, unless communication is extremely inexpensive. In Table 18 we show the additional costs involved in handling the corner points based on the simplest multiple scatter algorithm from [27]. The costs assigned are not necessarily optimal, but they are representative.

As can be seen from comparison with Tables 15 and 14, these local and global costs are comparable to or less than the costs for the local and global communication already present in the algorithm, provided that α/β is not less than $O(\sqrt{p})$ for the mesh and not less than $O(p/\log p)$ for the cube. The region of parameters we have investigated in this section lies up against these respective boundaries, but does not violate them severely. Therefore, solving the crosspoint system may roughly double the communication time when p is at the upper end of our range. This will be significant if communication is already expensive but it will not bring down the plateaus of high efficiency in Figures 12, 14 and 16. Thus, solving the corner point problem will not be a significant additional cost on machines already operating in favorable regions of parameter space.

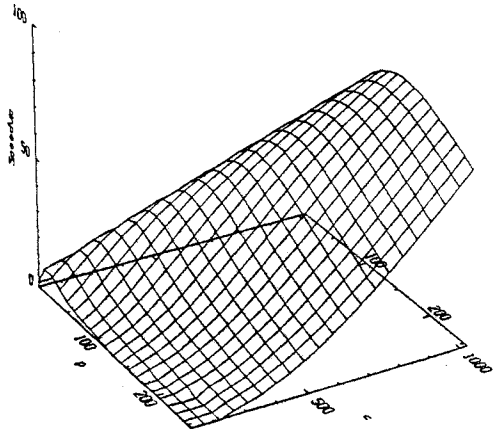
6. Conclusions

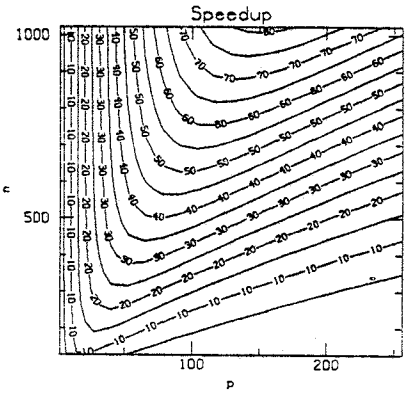
Our conclusions fall into two categories: those pertaining to domain decomposition algorithms themselves, and those pertaining to domain decomposition as an example of a general parallel computation.

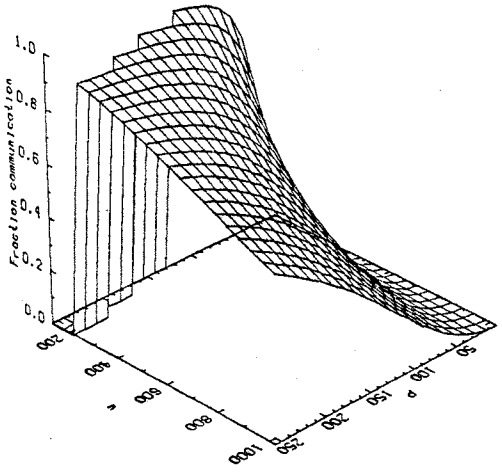
6.1. Domain Decomposition Algorithms

Two basic forms of the algorithm (SCM and PMM), different decomposition topologies, and many preconditioners for the separator set equations, each with advantages and disadvantages, make the study of domain decomposition surprisingly rich, since it is rooted in rather classical and straightforward ideas.

Because of its lower computational complexity, we prefer the SCM when it is applicable, namely when the subdomain solves can be handled exactly. In this case the full PMM iterates reduce to the SCM iterates and the latter can be obtained more cheaply. The PMM is a more flexible







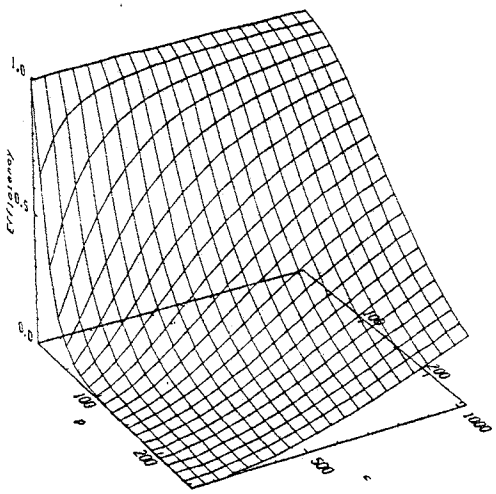
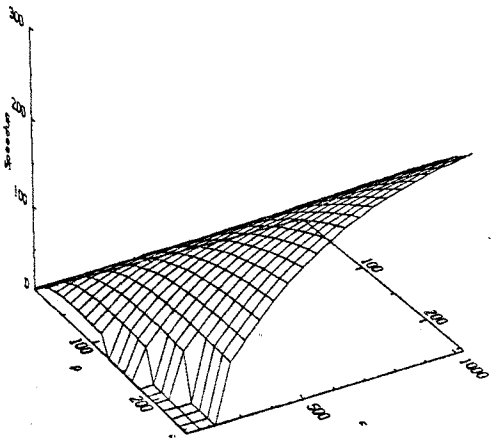
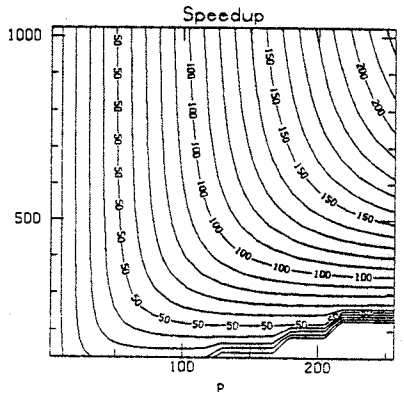
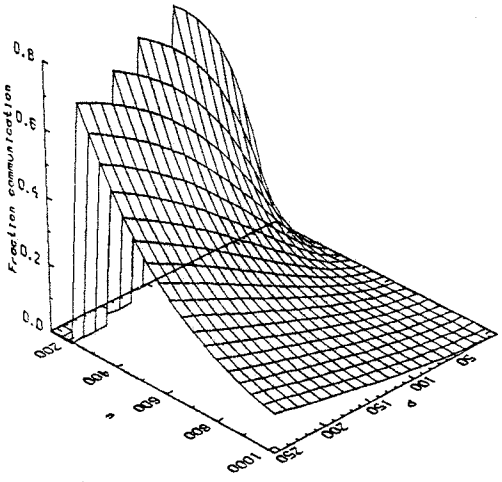


Figure 13: Theoretical parallelization measures for a ring, using strips for the domain decomposition. Communication-dominated processor.







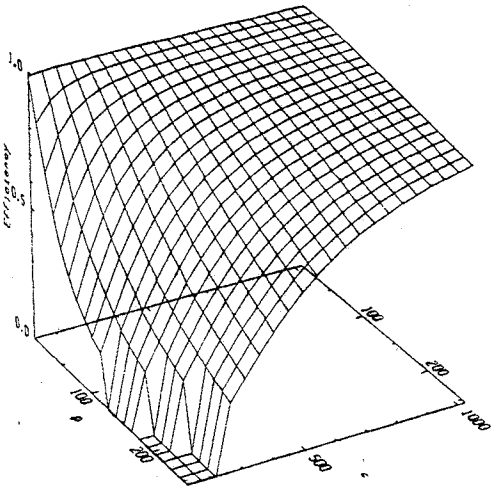


Figure 14: Theoretical parallelization measures for a mesh, using boxes for the domain decomposition. Communication-dominated processor.

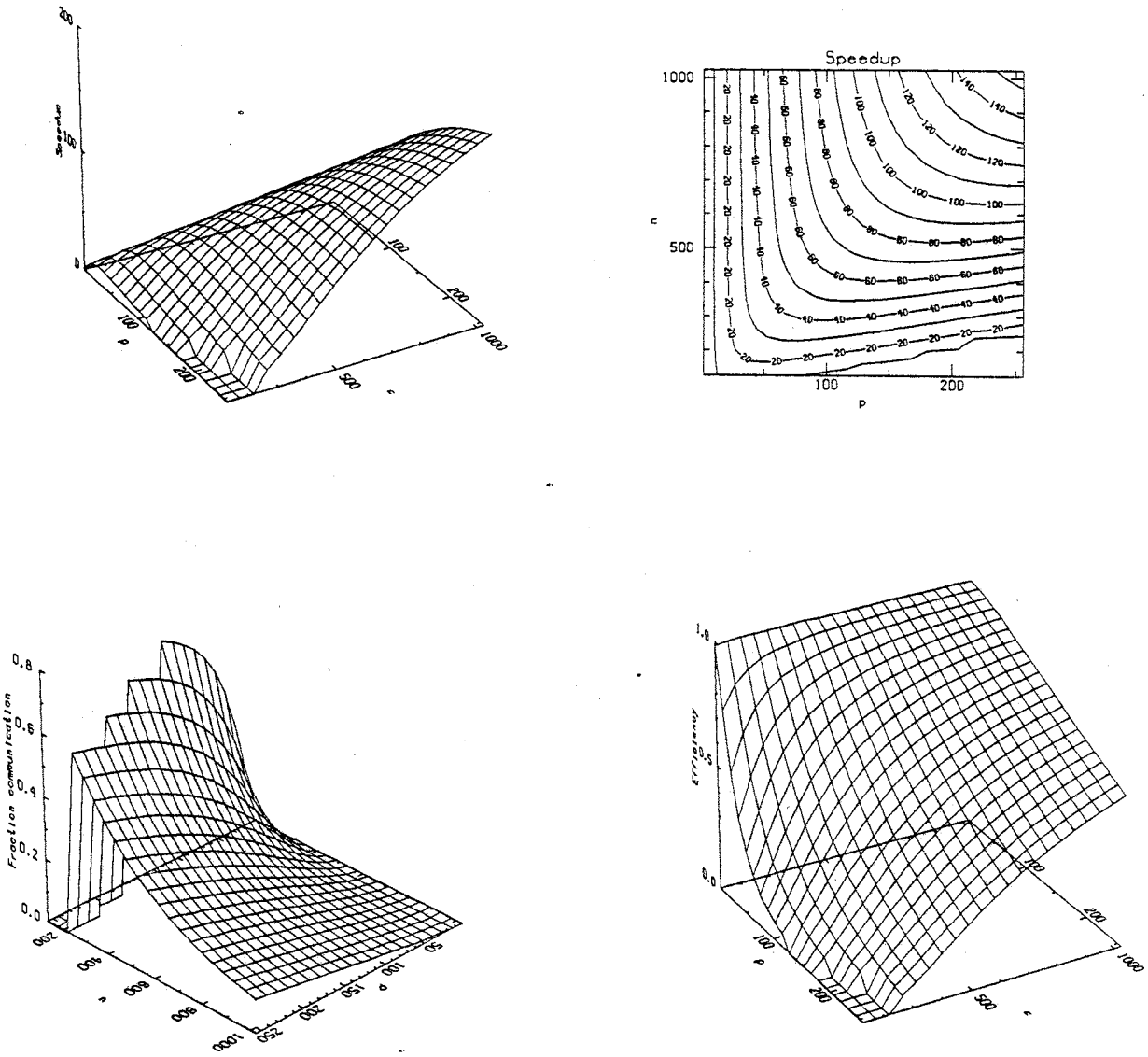
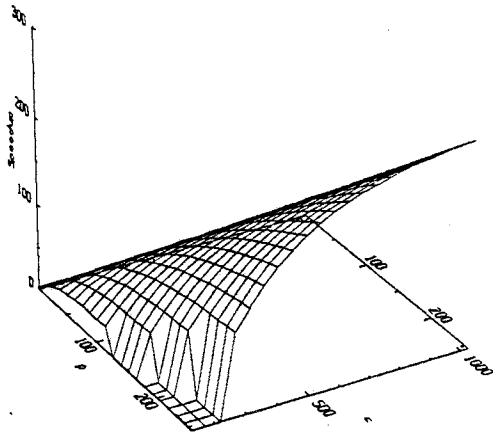
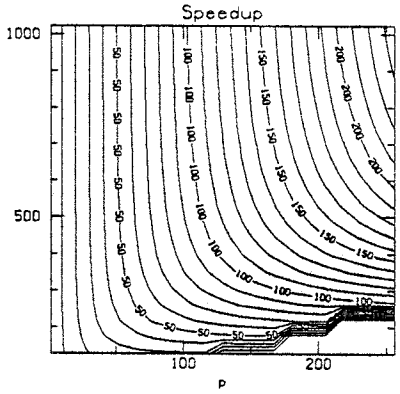
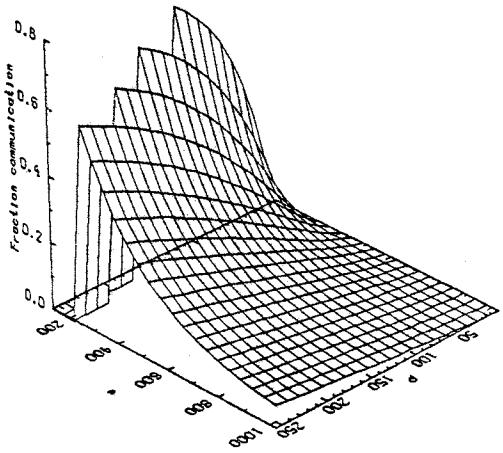


Figure 15: Theoretical parallelization measures for a hypercube, using strips for the domain decomposition. Communication-dominated processor.







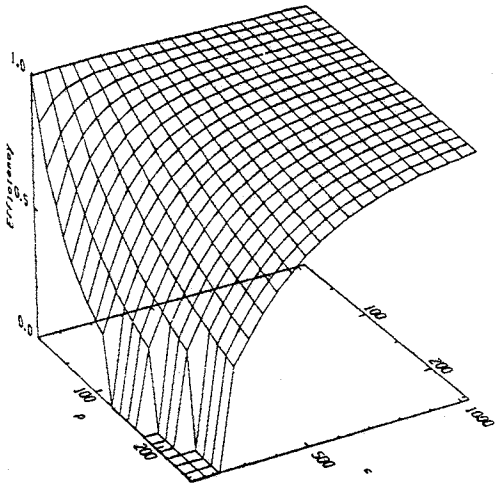


Figure 16: Theoretical parallelization measures for a hypercube, using boxes for the domain decomposition. Communication-dominated processor.

Comm. Type	Crosspoint Time
Assembly, either arch.	$2(\alpha+\beta)$
Scatter, 2-D Mesh	$2\alpha p^{\frac{1}{2}}+\beta(p^{\frac{1}{2}}+p)$
Scatter, Hypercube	$2\alpha \log p+\beta p$

Table 18: Communication times for the implicit handling of the crosspoint system.

algorithm for general symmetric scalar problems, but many generalizations are needed, especially to nonsymmetric systems of equations, where some generalized form of preconditioned conjugate gradients, such as the generalized minimum residual method, will come into play, and where the optimal preconditioners for the symmetric positive definite version of the algorithm will have less value.

The optimal preconditioners are to be prefered to the MSC preconditioners over the full range of examples contained herein, but the latter offer more possibilities for generalization and show at least some promise in dealing with coefficient irregularity.

In terms of convergence properties, boxwise decompositions have been demonstrated preferable to stripwise decompositions for large numbers of processors, and the crosspoint-coupled version of the algorithm is preferable from the point of view of iteration count.

Our parallelization analysis has revealed the important practical result that for reasonably achievable machine parameters and reasonably large problems, boxwise decompositions can be made more efficient than stripwise decompositions per iteration, in addition to offering better convergence properties; and that when domain decomposition is efficiently parallelizable at all, the crossoint-coupled version of the algorithm is not much less efficient than the decoupled version. This argues for further research emphasis in the direction of more implicit preconditioners.

6.2. Domain Decomposition as an example of parallel computation

The purpose of studying parallel computation is to not merely to invent and analyze new algorithms, but also to make suggestions about what types of parallel computers will be effective for scientific computing. Domain decomposition provides one set of non-trivial algorithms which test many features of parallel computers. An analysis of domain decomposition as a model of a general parallel computation reveals some interesting information about distributed memory parallel computer design. First, the time to start an I/O operation must not be too much larger than the time to actually send a single word. For the range of problems considered theoretically, the startup time α should not be more than an order of magnitude larger than the transfer time β. As shown in Figure 9, if the startup time is much larger than this, then global communication will dominate the communication and a different communication structure for the parallel computer may be more cost effective.

The "best" ratio of communication speed to computation speed is very dependent on the problem size and the decomposition granularity. The arithmetic complexity of elliptic PDEs is such that for moderate to coarse granularity communication will not dominate computation as long as communication speeds are comparable to the floating point speed. However, either a large startup time or a slow transfer rate will significantly degrade performance (cf. Figures 13–16). Figure 10 shows that both α and β may be an order of magnitude or so slower than f without making communication a dominant part of the cost. This result, with the results on α/β, argues for parallel processors with fast nodes and communication which may be a little slower than the computation. Therefore, engineering effort should not be spent making communication as fast as computation unless the cost of doing so is low.

In terms of architectures, our results argue in favor of hypercubes for domain decomposition. However, the advantage of the hypercube over the mesh for the boxwise decomposition can be slight in the region of parameter space where high efficiency is found, so if significant engineering improvements can be obtained at the expense of the richer interconnect of the hypercube, such a compromise is worth considering.

Our frustrating experiences in trying to make a relatively small program and its data area *fit* on a hypercube for problems of practical dimensions leads to a final comment concerning large-scale parallelism in scientific computing. In an MIMD computer, large amounts of memory are devoted to storing multiple images of the user's program. For example, in our tests, the program itself occupied about 200 Kbytes, *not counting* data areas. Since our test problem was idealized in many respects, it is easy to see that a genuine production program would need substantially more memory. In fact, 4 Megabytes per node might not be adequate. A 256 processor system with this much memory per node would have a total of 1 Gigabyte of memory, much of which would be used to store copies of the user's program and the operating system. This suggests that large scale parallel computers either follow a SIMD model or provide some way to share, perhaps in clusters, memory for the user's program and operating system.

7. Acknowledgements

We would like to thank all of the members of the numerical analysis community at Yale for their stimulation during our informal "Reading Group for Elliptic System Substructuring", which met during the spring of 1985. We are especially grateful for the contributions of Stan Eisenstat, Tony Chan, Diana Resasco, and Doug Baxter, which are only incompletely reflected in the bibliography.

References

[1] P. E. Bjorstad and O. B. Widlund, *Iterative Methods for the Solution of Elliptic Problems on Regions Partitioned into Substructures,* Technical Report 136, Courant Institute of Mathematical Sciences, NYU, September 1984.

[2] ————, Solving Elliptic Problems on Regions Partitioned into Substructures, G. Birkhoff and A. Schoenstadt ed., *Elliptic Problem Solvers II,* Academic, 1984, pp. 245–256.

[3] M. J. Berger and S. Bokhari, *A Partitioning Strategy for Non-uniform Problems on Multiprocessors,* Technical Report 85-55, ICASE, November 1985.

[4] J. H. Bramble, J. E. Pasciak and A. H. Schatz, *The Construction of Preconditioners for Elliptic Problems by Substructuring, I,* to appear in Math. Comp.

[5] ————, *An Iterative Method for Elliptic Problems on Regions Partitioned into Substructures,* to appear in Math. Comp.

[6] T. F. Chan, *Analysis of Preconditioners for Domain Decomposition,* Technical Report 408, Computer Science Dept., Yale University, August 1985.

[7] T. F. Chan and D. Resasco, *A Domain-Decomposed Fast Poisson Solver on a Rectangle,* Technical Report 409, Computer Science Dept., Yale University, October 1985.

[8] R. Chandra, *Conjugate Gradient Methods for Partial Differential Equations,* Technical Report YALEU/DCS/RR–129, Computer Science Dept., Yale University, January 1978.

[9] P. Concus, G.H. Golub and D.P. O'Leary, A Generalized Conjugate Gradient Method for the Numerical Solution of Elliptic Partial Differential Equations, J.R. Bunch and D.J. Rose ed., *Proceedings of the Symposium on Sparse Matrix Computations,* Academic Press, New York, 1975, pp. 309–332.

[10] Q. V. Dinh, R. Glowinski and J. Periaux, Solving Elliptic Problems by Domain Decomposition Methods with Applications, G. Birkhoff and A. Schoenstadt ed., *Elliptic Problem Solvers II,* Academic, 1984, pp. 395–426.

[11] M. Dryja and W. Proskurowski, Capacitance Matrix Method using Strips with Alternating Neumann and Dirichlet Boundary Conditions, R. Vichnevetsky and R. S. Stepleman ed., *Advances in Computer Methods for Partial Differential Equations - V,* IMACS, 1984, pp. 360–368.

[12] ————, *Capacitance Matrix Method using Strips with Alternating Neumann and Dirichlet Boundary Conditions,* Appl. Numer. Math., 1 (1985).

[13] M. Dryja, *A Capacitance Matrix Method for Dirichlet Problem on Polygonal Region,* Numer. Math., 39 (1982), pp. 51–64.

[14] ————, *A Finite Element – Capacitance Method for Elliptic Problems on Regions Partitioned into Subregions,* Numer. Math., 44 (1984), pp. 153–168.

[15] S. Eisenstat, *personal communication,* 1985.

[16] R. Glowinski, Q. V. Dinh and J. Periaux, *Domain Decomposition Methods for Nonlinear Problems in Fluid Dynamics,* to appear in Comp. Meths. Appl. Mech. Eng.

[17] G. H. Golub and D. Mayers, *The Use of Pre-Conditioning over Irregular Regions,* 1983. Lecture at Sixth Int. Conf. on Computing Methods in Applied Sciences and Engineering, Versailles, Dec. 1983.

[18] W. D. Gropp, *Dynamic Grid Manipulation for PDEs on Hypercube Parallel Processors,* in preparation.

[19] S. L. Johnsson, *Communication Efficient Basic Linear Algebra Computations on Hypercube Architectures,* Technical Report 361, Computer Science Department, Yale University, September 1985.

[20] C. Lanczos, *An Iteration Method for the Solution of the Eigenvalue Problem of linear differential and integral operators,* J. Res. NBS, 45 (1950), pp. 255–282.

[21] J. Oliger, W. Skamarock and W.-P. Tang, *Schwarz Alternating Procedure and S.O.R. Accelerations,* Technical Report, Computer Science Dept., Stanford University, 1985.

[22] A. T. Patera, *A Spectral Element Method for Fluid Dynamics: Laminar Flow in a Channel Expansion,* J. Comp. Phys., 54 (1984), pp. 468–488.

[23] J. S. Przemieniecki, *Matrix Structural Analysis of Substructures,* AIAA J., 1 (1963), pp. 138–147.

[24] D. Resasco, *personal communication,* 1985.

[25] J. R. Rice, E. N. Houstis and W. R. Dyksen, *A Population of Linear Second Order, Elliptic Partial Differential Equations on Rectangular Domains - Part I,* Technical Report 2078, Mathematics Research Center, Univ. of Wisconsin - Madison, May 1980.

[26] G. Rodrigue and J. Simon, Jacobi Splittings and the Method of Overlapping Domains for Solving Elliptic P.D.E.'s, R. Vichnevetsky and R. S. Stepleman ed., *Advances in Computer Methods for Partial Differential Equations - V,* IMACS, 1984, pp. 383–386.

[27] Y. Saad and M. Schultz, *Data Communication in Hypercubes,* Technical Report YALEU/DCS/RR–428, Computer Science Dept., Yale University, October 1985.

[28] H.A. Schwarz, *Gesammelte mathematische Abhandlungen,* Berlin, Springer, 2 (1890), pp. 133-134.

[29] P. N. Swarztrauber and R. A. Sweet, *Efficient FORTRAN Subprograms for the Solution of Separable Elliptic Partial Differential Equations,* ACM Trans. Math. Soft., 5 (1977), pp. 352–364.