Fast Parallel Direct Solvers For Coarse Grid Problems

H. M. Tufo[1] and P. F. Fischer[2]

Abstract

We develop a fast direct solver for parallel solution of "coarse grid" problems, $A\underline{x} = \underline{b}$, such as arise when domain decomposition or multigrid methods are applied to elliptic partial differential equations in d space dimensions. The approach is based upon a (quasi-) sparse factorization of the *inverse* of A. If A is $n \times n$ and the number of processors is P, the algorithm requires $O(n^\gamma \log P)$ time for communication and $O(n^{1+\gamma}/P)$ time for computation, where $\gamma \equiv \frac{d-1}{d}$. The method is particularly suited to leading edge multicomputer systems having thousands of processors. It achieves minimal message startup costs and substantially reduced message volume and arithmetic complexity compared to competing methods, which require $O(n \log P)$ time for communication and $O(n^{1+\gamma})$ or $O(n^2/P)$ time for computation. Timings on the Intel Paragon and ASCI-Red machines reflect these complexity estimates.

1 Introduction

In this paper we consider parallel direct solution methods for linear systems of the form

$$A\underline{x} = \underline{b}\,, \tag{1}$$

where A is an $n \times n$ sparse symmetric positive definite (SPD) matrix, such as arises from finite difference or finite element discretizations of d-dimensional elliptic partial differential equations (PDEs). We target our algorithm for the "fine-grained" regime, $n/P \approx 1$, and large number of processors, P. Moreover, we assume that $\underline{b}$ and and $\underline{x}$ are distributed vectors. Such problems are frequently encountered when computing coarse grid projections in the context of multigrid or domain decomposition solutions of much larger systems (e.g.[5, 6]). This study focuses on coarse grid *solution* times rather than factor times, as we expect to amortize the factorization costs over several iterations and/or time-steps of the larger governing systems. Although the local solves in domain decomposition are intrinsically parallel, the coarse grid problem (1) is a potential bottleneck, particularly for large P, because it effects global coupling and therefore requires global communication. Per force, the coarse grid data and solution before and after the solve stage are distributed, so it is not possible to consider solving the problem on fewer processors (artificially increasing n/P) without inducing additional communication overhead. (This point is discussed further in

[1]Department of Computer Science, The University of Chicago, 1100 East 58th Street, Ryerson 152 Chicago, IL 60637. *hmt@asci.uchicago.edu.*

[2]Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439. *fischer@mcs.anl.gov*

Section 6.) Although the algorithm we present can be efficiently implemented on a shared-memory architecture we limit our discussion to implementation on a distributed-memory multicomputer, that is, a collection of compute nodes having locally addressed private memory and connected by a communication network.

The problem of the coarse grid solve has been studied widely in the domain decomposition community. It has been established by Widlund [21] that order-independent convergence rates in domain decomposition methods cannot be obtained without the solution of a coarse grid problem. In [5], Chan and Shao present a study of the optimal coarse grid size for parallel applications which illustrates the importance of a fast coarse grid solver. Gropp *et al.* [12, 14, 18] also discuss the importance and challenge of developing an efficient parallel coarse grid solver for domain decomposition methods. Cai [4] has developed a domain decomposition scheme requiring a very low-dimensional coarse grid space where much of the information transfer is through the action of the restriction/prolongation operators. Nonetheless, it is clear that a multicomputer implementation of the coarse grid problem must require communication in the prolongation/restriction phase or have a minimum of one degree-of-freedom per processor, that is, $n \geq P$. Since leading edge multicomputer systems are currently scaling to thousands of processors, there is a clear need for an efficient treatment of the this problem. If the work per processor remains constant while the number of processors increases (the standard model for scaled speed up [15]), the coarse grid problem will ultimately dominate the complexity unless the solve time can be substantially reduced. The situation is worse in the fixed-problem-size case, where the local solution time scales as $1/P$ while the coarse grid solve time scales as $\log P$, due to the required global communication.

In this paper we discuss the implementation of a fast parallel coarse grid solution algorithm originally introduced in [9]. It is based upon creating a sparse A-conjugate basis for $\mathbb{R}^n$, to be denoted by the columns of the matrix

$$X = (\underline{x}_1, \ldots, \underline{x}_n)\,.$$

We show that this approach constitutes a sparse (not necessarily triangular) factorization of the full matrix A^{-1}. The scheme has a per-solve complexity of $O(n^\gamma \log P)$ for communication and $O(n^{1+\gamma}/P)$ for computation, where $\gamma \equiv \frac{d-1}{d}$. This compares quite favorably with more commonly used approaches which require $O(n \log P)$ time for communication and $O(n^{1+\gamma})$ or $O(n^2/P)$ time for computation. Results obtained on 512 nodes of an Intel Paragon and 2048 nodes of the Intel ASCI-Red machine at Sandia National Laboratories show that our method performs well even for values of n/P significantly greater than unity, particularly for large values of P.

We emphasize that in this paper we adopt a purely algebraic viewpoint because our intent is to solve (1) exactly, rather than to replace it with an approximate operator. We consider the potential of iterative approaches and extensions to approximate solutions in the closing discussion. Related ideas in the area of approximate factorizations can be found, for example, in [2, 3, 16].

The outline of the paper is as follows. Section 2 briefly describes several communication primitives central to distributed direct solution methods. Section 3 reviews several earlier

coarse grid solution strategies. Section 4 discusses the computational and communication complexity for the present approach. In Section 5 we present performance results for a $\sqrt{n} \times \sqrt{n}$ grid problem on the Intel Paragon and ASCI-Red platforms. Closing analysis and discussion are presented in Section 6.

2 Communication primitives

In this section, we review the communication complexity of several all-to-all communication schemes of relevance to coarse grid solvers.

If the originating PDE has an elliptic component, the inverse of A will be a full matrix. Consequently the coarse grid computation

$$\underline{x} = A^{-1}\underline{b}\,, \tag{2}$$

for distributed vectors $\underline{x}$ and $\underline{b}$ will require some type of all-to-all communication; any nonzero element of $\underline{b}$ will influence every element of $\underline{x}$. By definition, coarse grid problems are relatively fine-grained, implying that communication accounts for a substantial fraction of the solution time. Moreover, it is frequently the case that the messages are quite short implying that the communication phase is latency dominated, and hence, minimizing the total number of message startups is of paramount importance. If we assume that the compute nodes can receive only one message at a time, it follows that the minimum number of message cycles required to effect the requisite all-to-all communication in the evaluation of (2) is $\log_2 P$. *This is a lower bound on the communication complexity for solving (1).*

For communication time, we assume a standard linear model for contention-free data exchanges in which the time to send an m-word message is given by

$$t_c[m] = (\alpha + \beta m)t_a\,, \tag{3}$$

where α is the message startup cost (latency), β is the asymptotic per-word-transfer cost (inverse bandwidth), and t_a is the characteristic time for an arithmetic operation. Typically, $\alpha \gg \beta \gg 1$. We further assume that contention-free transit time is independent of the distance between processors (i.e., the processor network can be modeled as a switching network).

Our communication analysis in the forthcoming sections will be based upon the use of fan-in/fan-out reduction operations which are guaranteed to be contention-free, even on one-dimensional networks. To clarify this point, consider the implementation of two common reduction operations, vector concatenation and vector summation. The first gathers a distributed m-vector having $m_p = m/P$ components on each processor, $p = 0, 1, \ldots, P-1$. The second computes

$$\underline{v} = \sum_{p=0}^{P-1} \underline{v}^{(p)}\,,$$

where each $\underline{v}^{(p)}$ is an m-vector. Assuming $P = 2^D$, these algorithms can be described as follows:

Procedure Vector-Concatenate

Gather via binary fan-in

$\hat{m} := m_p$
do $l = 1$ to D
  if $mod\,(p, 2^l) = 0$ then
    recv $v(\hat{m}{+}1{:}2\hat{m})$ from $p + 2^{l-1}$
  elseif $mod\,(p, 2^{l-1}) = 0$ then
    send $v(1{:}\hat{m})$ to $p - 2^{l-1}$
  endif
  $\hat{m} := 2\hat{m}$
enddo

Broadcast via binary fan-out

do $l = D$ to 1 by -1
  if $mod\,(p, 2^l) = 0$ then
    send $v(1{:}m)$ to $p + 2^{l-1}$
  elseif $mod\,(p, 2^{l-1}) = 0$ then
    recv $v(1{:}m)$ from $p - 2^{l-1}$
  endif
enddo

Procedure Vector-Sum

Gather via binary fan-in

do $l = 1$ to D
  if $mod\,(p, 2^l) = 0$ then
    recv $w(1{:}m)$ from $p + 2^{l-1}$
    $v(1{:}m) := v(1{:}m) + w(1{:}m)$
  elseif $mod\,(p, 2^{l-1}) = 0$ then
    send $v(1{:}m)$ to $p - 2^{l-1}$
  endif
enddo

Broadcast via binary fan-out

do $l = D$ to 1 by -1
  if $mod\,(p, 2^l) = 0$ then
    send $v(1{:}m)$ to $p + 2^{l-1}$
  elseif $mod\,(p, 2^{l-1}) = 0$ then
    recv $v(1{:}m)$ from $p - 2^{l-1}$
  endif
enddo

In the fan-in stage, communication begins between neighboring processors, then neighbors of neighbors, etc., until in the last stage processor P/2 sends data to processor 0. In the case of a one-dimensional sequentially ordered network, or a multi-dimensional lexicographically ordered network, there will be no network contention because intermediate processors become inactive once their data is sent. The model (3) is therefore appropriate, and we estimate that the concatenate routine requires $(\alpha \log_2 P + \beta \cdot (m - m_p))t_a$ time for the gather, plus an additional $(\alpha + \beta m) \log_2 P\, t_a$ time for the broadcast, yielding a total time of $(2\alpha + \beta m) \log_2 P\, t_a$. The vector sum requires a total time of $(2\alpha + 2\beta m + m) \log_2 P\, t_a$, with the extra $m \log_2 P t_a$ term accounting for the vector summation in line four of the algorithm.

In contrast to the fan-in/fan-out strategy the respective operations can also be implemented via recursive doubling as follows:

Procedure Vector-Concatenate

Recursive Doubling

$\hat{m} := m_p$
do $l = 1$ to D
  send $v(1{:}\hat{m})$ to $mod\,(p + 2^{l-1}, P)$
  recv $v(\hat{m}{+}1{:}2\hat{m})$ from $mod\,(P{+}p{-}2^{l-1}, P)$
  $\hat{m} := 2\hat{m}$
enddo

Procedure Vector-Sum

Recursive Doubling

do $l = 1$ to D
  send $v(1{:}m)$ to $mod\,(p + 2^{l-1}, P)$
  recv $w(1{:}m)$ from $mod\,(P{+}p{-}2^{l-1}, P)$
  $v(1{:}m) := v(1{:}m) + w(1{:}m)$
enddo

In this case, the concatenate routine nominally requires time $(\alpha \log_2 P + \beta \cdot (m - m_p))t_a$, roughly a factor of $\log_2 P$ better than the fan-in/fan-out approach. The vector sum routine

requires a nominal time of $(\alpha + \beta m + m)\log_2 P\, t_a$, roughly a factor of two superior to the fan-in/fan-out approach. On hypercubes, the recursive doubling algorithms can be implemented with a contention-free schedule. However, on low-dimensional networks the recursive doubling schemes will suffer observable network contention unless the problem is latency dominated, that is, $m \lesssim \alpha/\beta$. Consequently, the performance will generally be inferior to the fan-in/fan-out approach.

It is worth noting that hybrid approaches are possible. For example, for concatenation, recursive doubling can be used until $\hat{m} \approx \alpha/\beta$ and fan-in/fan-out then used on processor subsets to span the remaining levels of the tree(s). For vector summation there is a well known hybrid scheme due to van de Geijn *et al.* [20] which is effective when $m \gg \alpha/\beta$. For the values of m required for the coarse grid solution schemes considered here, the fan-in/fan-out schemes capture the essential complexity. Hybrid schemes would be most appropriate as fine tuning measures in the final implementation phases and we do not consider them further.

To put the forthcoming discussion of solution strategies on a firm foundation, we briefly review the communication requirements of a typical matrix-vector multiplication implementation on $P = 2^D$ processors. Assume $\underline{x}$ is a vector having n components, $(x_1, \ldots, x_{i_g}, \ldots, x_n)$, which are distributed across P processors according to the bijective map, $i_g = \mu(i,p) \in \{1, \ldots, n\}$, where $i \in \{1, \ldots, n_p\}$, is the local index on processor p, for $p \in \{0, \ldots, P-1\}$. The global-to-local mapping is specified by the inverse mapping, $(i,p) = \mu^{-1}(i_g)$. We assume, without loss of generality, that the number of components of $\underline{x}$ on each processor p is the same, that is, $n_p = n/P$. Let $C = (\underline{c}_1\, \underline{c}_2 \ldots \underline{c}_n)$ be an $n \times n$ matrix with each column, $\underline{c}_i$, partitioned according to the same distribution as $\underline{x}$ (i.e., *rows* of C are contiguous within a processor, with row $\mu(i,p)$ of C mapped to row i on processor p). Then $\underline{y} = C\underline{x}$ is computed as follows. First, a copy of $\underline{x}$ is gathered onto each processor using a vector-concatenate procedure. Then, each processor p computes the inner-products $y_{\mu(i,p)} = \underline{x}^T \underline{r}_{\mu(i,p)}$ for $i = 1, \ldots, n_p$, where $\underline{r}_\mu$ is the μth row of C. The end result is a vector $\underline{y}$, distributed according to the map μ. If C is full, the matrix-vector product complexity is $2n \cdot n_p t_a = 2(n^2/P)t_a$ for the local inner-products plus $(2\alpha + \beta m)\log_2 P\, t_a$ for the gather.

3 Survey of coarse grid solvers

It is well known (e.g., [12]) that parallel solution of the coarse grid problem is hampered by the inherent sequentiality of the forward and backward substitution phases of standard triangular (LU or LL^T) solves. If n (and consequently, P) is sufficiently small, it is feasible to store, factor, and solve the system locally within a single processor, thus allowing the use of standard serial solvers. On a low-dimensional network, the optimal variant of this scheme is to concatenate $\underline{b}$ via the binary fan-in scheme of the previous section, solve the problem on the root, and then cascade the solution from the root using the inverse of the concatenation procedure. If the local solution strategy is based upon banded solvers, the computational complexity is $4ns$ operations for a matrix of bandwidth s, while the communication complexity is $2\alpha \log_2 P + 2\beta n$, as noted in the previous section. For historical reasons, it is more common to solve the problem redundantly on each processor, obviating

the need to broadcast the solution. On a hypercube, such a strategy is sensible because the recursive doubling variant of concatenation is contention free and the communication cost is halved. However, on lower-dimensional networks, the optimal communication strategy for the redundant solution approach is based upon fan-in/fan-out, with a cost of $(2\alpha + \beta n)\log_2 P\, t_a$.

For large numbers of processors and relatively small systems (e.g., $P > 128$, $n < 5000$), computing the full inverse of A can be far more effective than solving the system redundantly (e.g., [10, 13]). By distributing the rows of A^{-1} in the same manner as $\underline{x}$ and $\underline{b}$, the solution can be computed as a parallel matrix-vector product, $A^{-1}\underline{b}$, once $\underline{b}$ has been gathered onto each processor. The communication complexity is identical to that of the redundant LU method described above; however, the complexity for the computation of the inner-products of the rows of A^{-1} with $\underline{b}$ is $2n^2/P$. Parallelism has been introduced to this phase of the solution and it follows that the distributed A^{-1} approach is superior whenever $P > \frac{n}{2s}$.

The advantage of the distributed A^{-1} method is that matrix-vector multiplication is intrinsically parallel. Unfortunately, A^{-1} is completely full and, consequently, the storage cost of n^2/P per processor limits this approach to values of n of up to only a few thousand in practice. With the advent of computers containing thousands of processors this restriction is problematic. Ideally, one would like a matrix-vector product based approach involving *sparse* matrices.

A step in this direction is the method of Alvarado *et al.* [1] who develop fast parallel triangular solvers by recasting the inverse of a sparse triangular matrix, L, as a product of l sparse factors, $\tilde{L}_i^{-1}$, each of which can be computed in place. The solution for a single triangular system is then given by the sequence of products $\underline{v}_0 = \underline{b}$, $\underline{v}_i = \tilde{L}_i^{-1}\underline{v}_{i-1}, \ldots, \underline{x} = \tilde{L}_l^{-1}\underline{v}_{l-1}$. Analysis of this approach is quite difficult, since each factor is sparse, and it's unclear where data is located at the start and finish of each multiplication. However, Alvarado *et al.* strive to minimize l, in which case, each matrix-vector product must be performed in turn, with communication taking place in between (otherwise, there would be further parallelism to be exploited, and l would therefore not be minimal). Assuming that the work of each matrix-vector product is distributed across P processors, we estimate the communication time for each of the l cycles as $t_c = 2\alpha(\log_2 P)t_a$. If A is the discrete Laplacian for a problem on a two-dimensional grid, the number of factors is typically $l \approx \log_2\sqrt{n} \approx \log_2\sqrt{P}$ (see, e.g., [11]). Since solution of (1) requires both a forward- and backward-sweep, we estimate a lower bound on the solution time of $2\alpha(\log_2 P)^2 t_a$. This estimate neglects both the work, which, with a lower bound of $\Omega(\frac{n\log n}{P})$ [11], probably is negligible, and the amount of data communicated, which probably is not negligible.

Another approach of interest is that of Farhat and Chen [7] who solve the coarse grid problem by projecting onto sets of previously generated Krylov vectors that constitute an approximation space. Let $X_k = (\underline{x}_1\ \underline{x}_2\ \ldots\ \underline{x}_k)$ be a matrix of A-conjugate vectors normalized to satisfy

$$\underline{x}_i^T A \underline{x}_j \;=\; \delta_{ij}\,, \tag{4}$$

where δ_{ij} is the Kronecker delta. Then

$$\bar{\underline{x}} = X_k X_k^T \underline{b} \tag{5}$$

yields the projection onto $\mathcal{R}(X_k)$ satisfying

$$\bar{\underline{x}} \in \mathcal{R}(X_k)\,, \qquad ||\underline{x} - \bar{\underline{x}}||_A \leq ||\underline{x} - \underline{v}||_A \quad , \quad \forall \underline{v} \in \mathcal{R}(X_k)\,. \tag{6}$$

Here, $\mathcal{R}(\cdot)$ denotes the range of the argument and $||\cdot||_A$ denotes the A-norm given by $||\underline{w}||_A = (\underline{w}^T A \underline{w})^{\frac{1}{2}}$.

Farhat and Chen build the space $\mathcal{R}(X_k)$ by collecting the A-conjugate search directions generated in the course of applying (a slightly modified) conjugate gradient (CG) iteration to (1) for several right-hand sides. In time transient problems, the successive right-hand sides often share enough information such that very few CG iterations are required to solve the problem subsequent to the initial projection (6). In the examples reported in [7], Farhat and Chen observe that superconvergence sets in for $k \gtrsim 0.25n$, at which point only one or two conjugate gradient iterations are required subsequent to the initial projection.

We can estimate the complexity of the projection+CG approach by computing the cost of the projection step (though the subsequent CG iteration cost is in fact non-negligible). Assume that each basis vector, $\underline{x}_j$, is distributed in the same fashion as $\underline{x}$ and $\underline{b}$. To compute $\bar{\underline{x}} = X_k X_k^T \underline{b}$, one first computes an intermediate k-vector, $\underline{c} = X^T \underline{b}$, in two stages, beginning with evaluation of the local dot products

$$\begin{aligned} c_j^{(p)} \quad &= \quad \sum_{i=1}^{n_p} b_{\mu(i,p)}\, x_{\mu(i,p),j}\,, \qquad & j \in 1,\ldots,k \\ & & p \in 0,\ldots,P-1\,, \end{aligned} \tag{7}$$

followed by a $\log_2 P$ sum across processors

$$c_j \quad = \quad \sum_{p=0}^{P-1} c_j^{(p)} \qquad j \in 1,\ldots,k\,. \tag{8}$$

With the components of $\underline{c}$ known to every processor, the distributed vector $\bar{\underline{x}}$ is computed as

$$\begin{aligned} \bar{x}_{\mu(i,p)} \quad &= \quad \sum_{j=1}^{k} c_j x_{\mu(i,p),j} \qquad & i \in 1,\ldots,n_p \\ & & p \in 0,\ldots,P-1\,. \end{aligned} \tag{9}$$

This final stage is recognized as a sequence of k DAXPYs, $\bar{\underline{x}}^{(p)} = \bar{\underline{x}}^{(p)} + c_j \underline{x}_j^{(p)}$, of length n_p on each processor p, and is fully concurrent.

If the vectors $\underline{x}_j$ are full, this approximation has leading order computational complexity of $4nk/P$ operations for the required dot products (7) and DAXPYs (9). The communication time for the gather of the k coefficients of each column vector (8) is $\log_2 P(2\alpha + 2\beta k + k)t_a$, where the last k term accounts for the summation in (8). Note that if $k = O(n)$, then the projection approach is better than the distributed A^{-1} approach by at most a constant, with a lower bound solution time of $2\alpha t_a \log_2 P$ being obtained for both methods. Furthermore, any CG iterations required for the projection+CG scheme will incur additional latency overhead of at least $2\alpha t_a \log_2 P$ per iteration due to the inner-products required for the CG algorithm.

In the next section, we present a projection method for which $k \equiv n$ but which, by virtue of using a *sparse* basis set, X, achieves communication and computation complexities which are of lower order than the A^{-1} approach. Moreover, this approach requires

a minimum number of message cycles and thus achieves the lower bound latency time of $2\alpha t_a \log_2 P$.

4 Sparse basis projection method

The goal of the method of Farhat and Chen [7] is to choose a basis set X_k such that $\bar{\underline{x}}$ is a good approximation to $\underline{x}$. We observe that if $k = n$, then $\mathcal{R}(X_k) = \mathbb{R}^n$ and, from (6), $\bar{\underline{x}} \equiv \underline{x}$, implying that $X_n X_n^T$ is the inverse of A. In [9], we introduced a method in which the projection approach is modified to incorporate a matrix of n basis vectors, $X \equiv X_n$, which is as sparse as possible and which yields significantly reduced computational and communication complexities. Here, we describe the implementation of the method and discuss communication considerations in depth. In the next section, we compare performance of the XX^T-based method to the redundant LU and A^{-1} approaches of Section 3.

4.1 Basis

We begin with the following observation. Let the unit vectors $\hat{\underline{e}}_i$ and $\hat{\underline{e}}_j$ denote the ith and jth column of the $n \times n$ identity matrix. Let $\mathcal{N}_j$, the *neighborhood* of j, be the set of row

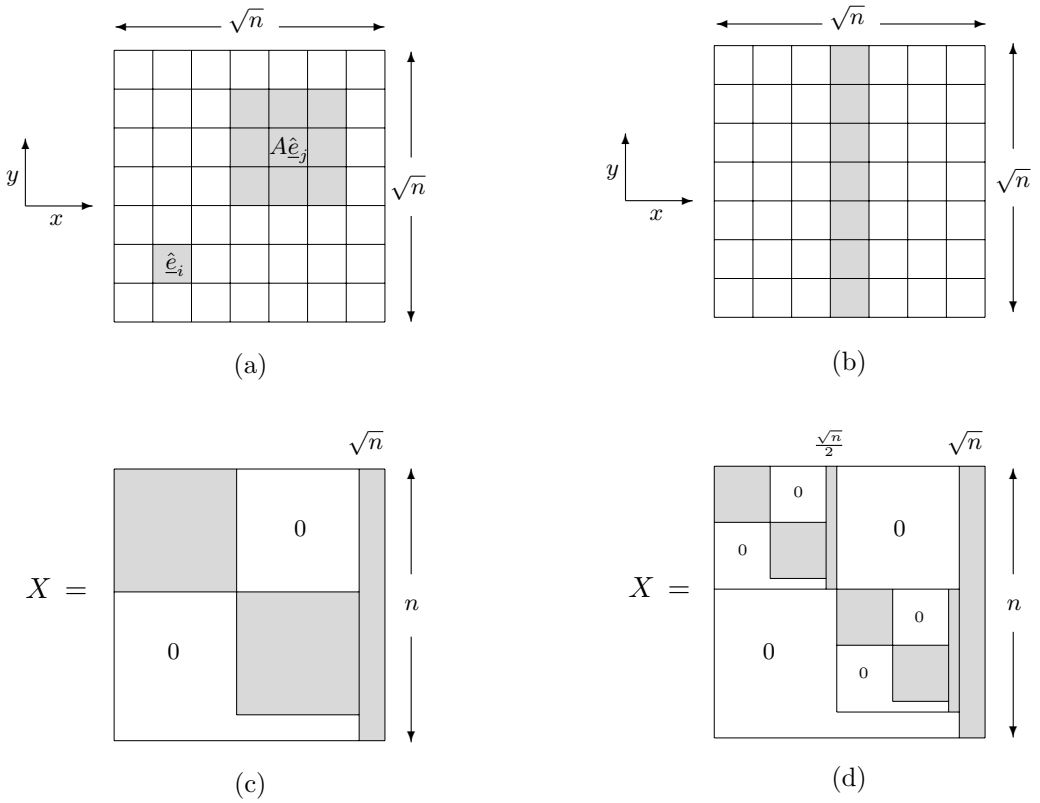


Figure 1: (a) geometric support (shaded) of orthogonal vectors $\hat{\underline{e}}_j$ and $A\hat{\underline{e}}_i$. (b) support of separator set. (c) zero/fill structure for X resulting from ordering the separator set last. (d) zero/fill structure after second round of recursion.

indices corresponding to nonzeros in column j of A, that is, $i \in \mathcal{N}_j$ *iff* $a_{ij} \neq 0$. Then

$$\hat{\underline{e}}_i^T A \hat{\underline{e}}_j \quad = \quad 0 \qquad \forall\, i \notin \mathcal{N}_j \quad .$$

This situation is illustrated in Fig. 1a for the case where A arises from a 9-point discretization. (In this and subsequent figures, the degrees-of-freedom are associated with the centroids of the cells in the computational grid.) From this figure it is clear that at least $n/\max|\mathcal{N}_j|$ of the unit vectors are A-conjugate to one another, where $|\mathcal{N}_j|$ denotes the cardinality of $\mathcal{N}_j$.

The generation of a sparse basis for X starts with finding a maximal (or near-maximal) set of k_1 A-conjugate unit vectors and normalizing them to satisfy (4). The first such k_1 columns of X will each have *only one nonzero entry.* Additional entries in X are generated via Gram-Schmidt orthogonalization. Let $X_k = (X_{k-1}\, \underline{x}_k)$ denote the $n \times k$ matrix with columns $(\underline{x}_1\, \underline{x}_2 \ldots \underline{x}_k)$, and let $V = (\underline{v}_1\, \underline{v}_2 \ldots \underline{v}_n)$ be an appropriate column permutation of the identity matrix. Then the procedure

$$\begin{aligned} &\textit{do}\ \ k = 1, \ldots, n\text{:} \\ &\quad \underline{w} := \underline{v}_k \ - \ X_{k-1} X_{k-1}^T A \underline{v}_k \\ &\quad \underline{x}_k := \underline{w} / ||\underline{w}||_A \\ &\quad X_k := (X_{k-1}\ \underline{x}_k) \\ &\textit{enddo} \end{aligned} \tag{10}$$

ensures that $X = X_n$ is the desired factor of A^{-1}. For $k \leq k_1$ the projection, $X_{k-1} X_{k-1}^T A \underline{v}_k$, computed in (10) will be void and $\underline{x}_k$ will simply be a multiple of $\underline{v}_k$. As k increases beyond k_1, X_k will begin to fill in. The goal is to find an ordering, V, which yields minimal or near minimal fill for the factor X.

Following [11], an efficient procedure for selecting the permutation matrix, V, can be developed by defining separators which recursively divide the domain (or graph) associated with A into nearly equal subdomains. Figure 1b shows the first such separator for a $\sqrt{n} \times \sqrt{n}$ grid. Since the stencil for $A\hat{\underline{e}}_j$ does not cross the separator, it is clear that every unit vector $\hat{\underline{e}}_i$ associated with the left half of the domain in Fig. 1b is A-conjugate to every unit vector $\hat{\underline{e}}_j$ associated with the right half. If V is arranged such that vectors associated with the left half of the domain are ordered first, vectors associated with the right half second, and vectors associated with separator last, then application of Gram-Schmidt orthogonalization will generate a matrix X with worst-case fill depicted by Fig. 1c (X is shown here with the rows ordered according to the same permutation used for the columns of V). This procedure can be repeated to order the vectors within each subdomain, giving rise to the structure shown in Fig. 1d. To complete the construction we recur until no more separators can be found.

It is clear from (5) that the computational complexity of each solve is proportional to the amount of nonzero fill in the factor X. For the $\sqrt{n} \times \sqrt{n}$ grid we observe from Fig. 1d that the number of nonzeros in each row is bounded by the sequence

$$\sqrt{n} + \frac{\sqrt{n}}{2} + \frac{\sqrt{n}}{2} + \frac{\sqrt{n}}{4} + \frac{\sqrt{n}}{4} + \ldots \ \leq 3\sqrt{n}\,,$$

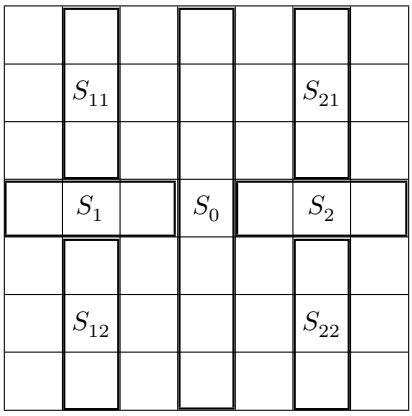
implying a total bound on the amount of fill in X of $3n\sqrt{n}$. Since we can evenly distribute the work among processors this leads to a computational complexity of $O(n^{\frac{3}{2}}/P)$. Similar arguments in three-dimensions lead to a computational complexity of $O(n^{\frac{5}{3}}/P)$. Both the two and three-dimensional cases provide a clear gain over the $O(n^2/P)$ cost incurred by the full inverse approach.

The communication complexity is dependent upon the mapping of the rows of X (and hence, $\underline{x}$ and $\underline{b}$) to the processors. In the worst case, the bound is simply that derived for (8), namely, $\log_2 P(2\alpha+2\beta n+n)t_a$, which is essentially the same as the A^{-1} approach. Because of the significant reduction in computational complexity, even a naive implementation of the XX^T approach will be superior to the A^{-1} approach. However, for properly mapped two-dimensional problems, it is possible to obtain a contention-free communication complexity bound of $(2\alpha \log_2 P + O(n^{\frac{1}{2}})\beta \log_2 P)t_a$, even on a linear array of processors. The three-dimensional bound is $(2\alpha \log_2 P + O(n^{\frac{2}{3}})\beta \log_2 P)t_a$. Results in Section 5 show that, for large problems, this lower communication complexity is as significant as the improved computational complexity in reducing the overall solution time.

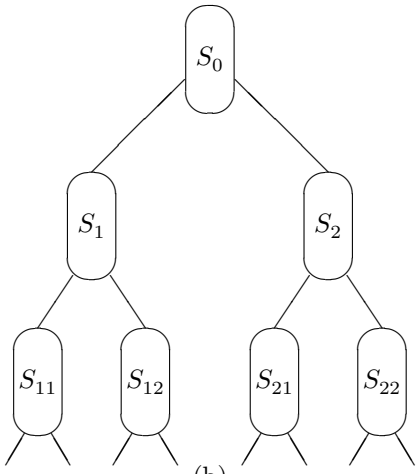
4.2 A detailed example

To understand the ordering and processor mapping requirements necessary to reduce the communication complexity we consider the 7×7 grid example of Fig. 2 in some detail. As in Fig. 1, the degrees-of-freedom are represented by the square cells shown in (a), and it is assumed that A has a 3×3 stencil. The first three levels of separators have been labeled in (a) and an associated hierarchy is depicted by the binary tree in (b). To obtain the desired nonzero structure of X, the separator labeling is continued until all degrees-of-freedom have been identified as an element of a separator. The degrees-of-freedom are then labeled in reverse order, $i = n, \ldots, 1$, using a depth-first traversal of the tree. One begins with elements in separator S_0, followed by those in S_2, S_{22}, and so on, to yield the orderings shown in (c) and (d). The descendants of an element j are denoted as the set $\mathcal{D}_j$ comprising j and any element i which is below j in the tree.

The Gram-Schmidt procedure (10) does not require the rows of X to be permuted with the same ordering as the columns. However, there are notational and implementation advantages to doing so. Thus, from here on we assume that A has been constructed according to the ordering in Fig. 2c, corresponding to a symmetric permutation of the original operator. In this case, the Gram-Schmidt procedure (10) is simplified in that the basis vectors become $\underline{v}_k = \underline{\hat{e}}_k$, $k = 1, \ldots, n$. Because of the reversed depth-first ordering, this corresponds to starting at the leaves of the tree (not shown in Fig. 2d); subsequent unit vectors are selected for orthogonalization only after their descendants. By construction, a unit vector $\underline{\hat{e}}_k$ is automatically A-conjugate to any unit vector which is not its direct descendant or direct ancestor. Hence, the Gram-Schmidt projection step only needs to be effected against the columns of X_{k-1} that correspond to descendants of k. Thus, (10) is recast as



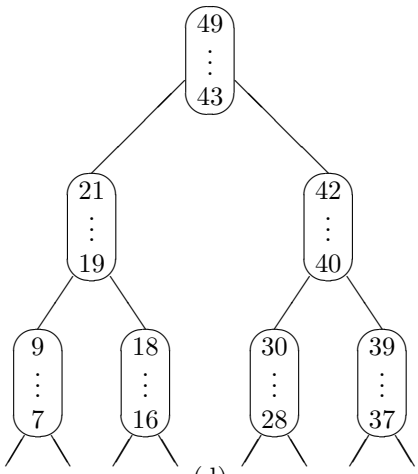
(a)



(b)

2	9	5	49	23	30	26
3	8	6	48	24	29	27
1	7	4	47	22	28	25
19	20	21	46	40	41	42
11	18	14	45	32	39	35
12	17	15	44	33	38	36
10	16	13	43	31	37	34

(c)



(d)

Figure 2: (a) separator sets (only first three levels shown). (b) separator set tree. (c) global numbering induced by depth first ordering of separator sets. (d) global numbering embedded in separator set tree.

$$
\begin{aligned}
&do \;\; k = 1, \ldots, n\text{:} \\
&\quad \underline{w}_k := \hat{\underline{e}}_k \;\; - \sum_{j \in \mathcal{D}_k \backslash k} \underline{x}_j (\underline{x}_j^T A \hat{\underline{e}}_k) \qquad (11) \\
&\quad \underline{x}_k := \underline{w}_k / ||\underline{w}_k||_A \\
&\quad X_k := (X_{k-1} \; \underline{x}_k) \\
&enddo
\end{aligned}
$$

In general, x_{ik} will be nonzero for all elements $i \in \mathcal{D}_k$, save the possibility of fortuitous cancellation during the projection step.

For the important case when A is an M-matrix, that is, SPD with non-positive off-diagonal entries, then it is guaranteed that all of the entries in X are non-negative and that there will be no cancellation during the projection step (11). The non-negativity of the x_{ij}'s

is established by induction. It clearly holds for the leaves of the tree since, in that case, each $\underline{x}_j$ is simply a positive multiple of $\hat{\underline{e}}_j$. Now consider the sign of the basis coefficients in the projection step (11):

$$\begin{aligned} \beta_j &= \underline{x}_j^T A \hat{\underline{e}}_k \qquad j \in \{1, ..., k-1\} \\ &= \sum_{i=1}^{n} x_{ij} a_{ik} \,. \end{aligned} \tag{12}$$

Since $k > j$, we have $x_{kj} = 0$, and all terms in the summation (12) are nonpositive by the assumptions $x_{ij} \geq 0$ and $a_{ij} \leq 0$, $i \neq j$. Therefore, $\beta_j \leq 0$. Since the vector $\beta_j \underline{x}_j$ is subtracted from from $\hat{\underline{e}}_k$ in (11), all elements of $\underline{w}_k$ and hence, $\underline{x}_k$, must be positive. It is interesting to note that simply adding positive components to the unit vector $\hat{\underline{e}}_k$ yields a vector ($\underline{w}_k$) having a greater 2-norm, but reduced A-norm. We conclude that this must result from "smoothing" the Kronecker delta function represented by $\hat{\underline{e}}_k$. Indeed, for the case when A is a discrete Laplacian, plots of the element distribution on the physical mesh reveal that the basis vectors $\underline{x}_k$ (or $\underline{w}_k$) smoothly decay away from element k to the boundary of the support of $\mathcal{D}_k$.

From the above arguments, we see that the number of nonzeros in any column $\underline{x}_j$ is (generally) going to be equal to $|\mathcal{D}_j|$, that is, the number of descendants of j. It follows that the number of nonzeros in a given row, i, is given by the number of ancestors of i, that is, by counting up from the location of i to the root of the tree. For example, in Fig. 2d, the number of nonzeros in column 49 of X will be 49, whereas the number of nonzeros in row 49 will be 1. We conclude that, thus generated, X is upper-triangular and therefore the unique Cholesky factor of A^{-1}.

To illustrate the significance of obtaining a proper ordering prior to generating X, we close this section with a one-dimensional example. Figs. 3a and b, show the sparsity patterns obtained for the upper-triangular Cholesky factor, L^T, and its inverse when A is the well-known tridiagonal matrix, $LL^T = A =$*tridiag* (-1,2,-1), deriving from a centered difference approximation to a second-order derivative. Despite the fact that L^T has the minimum possible fill, $(L^T)^{-1}$ is completely full. However, if one first permutes A using a depth-first nested-dissection ordering, V, and then computes the factors $L_N L_N^T = V^T A V$, one obtains the sparsity patterns shown in Figs. 3c and d. It is readily shown for this case that the number of nonzeros in $X \equiv (L_N^T)^{-1}$ is $O(n \log n)$. Of course, because the Green's function for the associated continuous equation is non-vanishing everywhere, $XX^T = (VAV^T)^{-1}$ is completely full.

4.3 Parallel Implementation

We now examine the influence of the nonzero pattern of X on the communication requirements for the parallel solver and show that this can be exploited to obtain a communication complexity which is significantly less than $O(n)$.

Recall that the nonzeros in each column $\underline{x}_j$ correspond to descendants of j in the separator tree, that is,

$$x_{ij} \neq 0 \implies i \in \mathcal{D}_j \,.$$

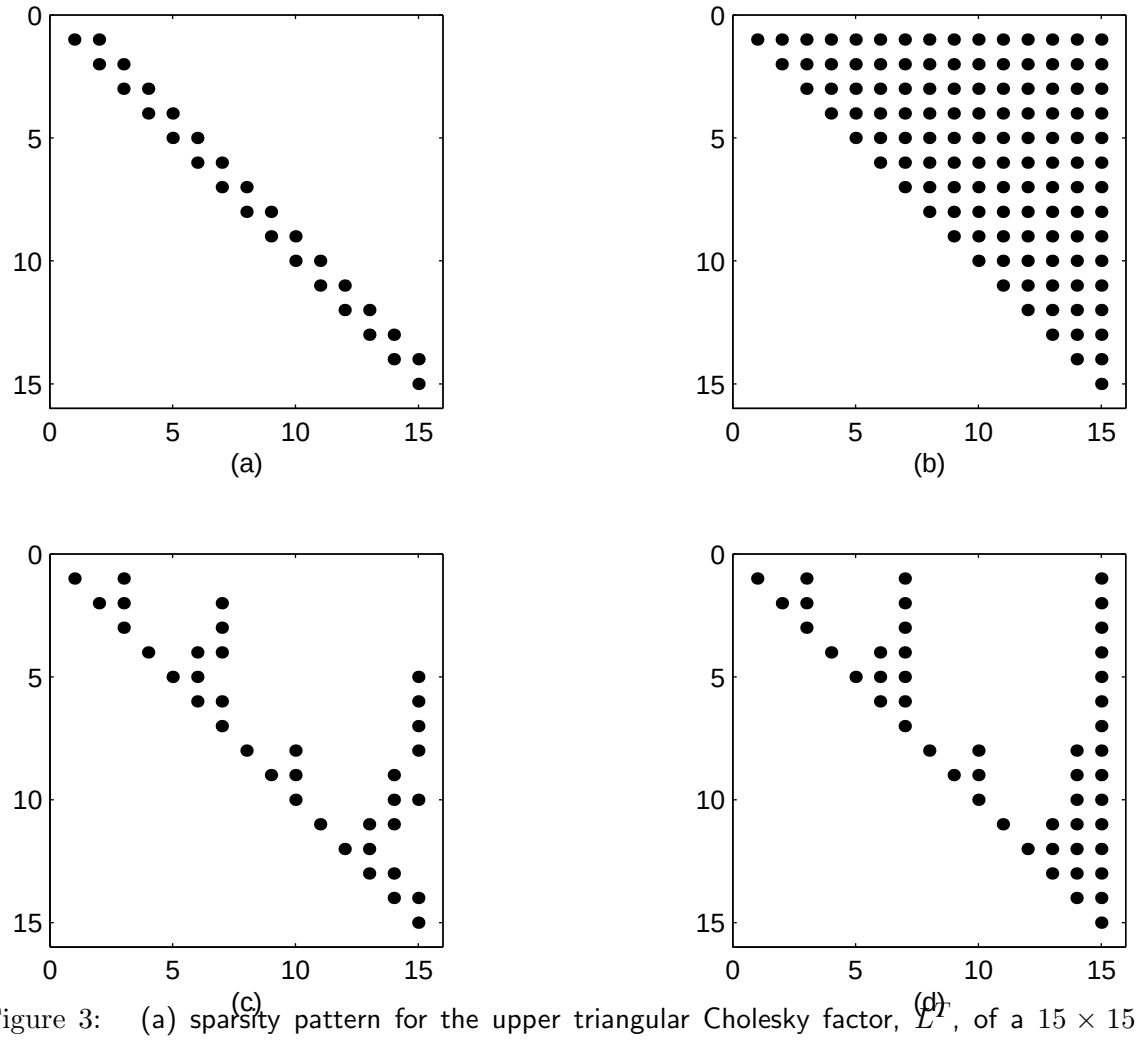


Figure 3: (a) sparsity pattern for the upper triangular Cholesky factor, L^T, of a 15×15 tridiagonal matrix, (b) sparsity pattern of $(L^T)^{-1}$, (c) sparsity pattern for the Cholesky factor, L_N^T, obtained from a nested dissection ordering of A, (d) sparsity pattern for $X = (L_N^T)^{-1}$.

Thus, the dependency graph in Fig. 2b reflects the input requirements for the evaluation of the dot products (7) and DAXPYs (9) during the computation of $XX^T\underline{b}$. The dot products, $c_j = \underline{x}_j^T\underline{b}$, are computed as

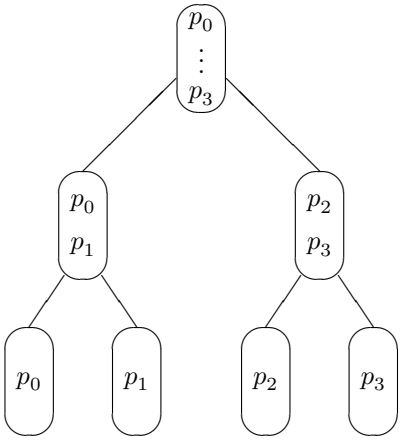
$$\begin{aligned} c_j &= \sum_{i=1}^{n} x_{ij} b_i \\ &= \sum_{x_{ij}\neq 0} x_{ij} b_i \\ &= \sum_{i\in\mathcal{D}_j} x_{ij} b_i \,, \end{aligned} \tag{13}$$

from which it is clear that the computation of c_j depends only upon the descendants of j.

Communication can be minimized during the computation of c_j (13) if the lower branches of the tree are, to the extent possible, self-contained within a given processor. This can be achieved in a natural way by assigning the processor distribution during the nested dissection phase of the ordering. Degrees-of-freedom to the left of the first separator

are assigned to the lower half-set of processors, those to the right are assigned to the upper half, and those belonging to the separator itself can be assigned to any processor in the set. Repeating this procedure recursively for the example problem of Fig. 2 leads to the element-to-processor distribution shown in Figs. 4a and b.

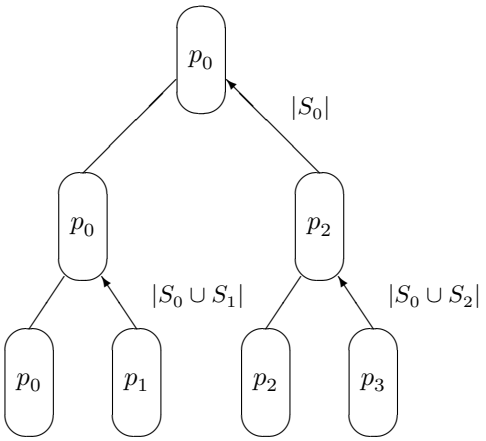
In general, one obtains an admissible element-to-processor map by simply overlaying the processor and separator trees. At each level, the elements of a given separator can be assigned to any processors in the pool associated with that branch of the tree. In the event that the processor tree has insufficient depth to cover the separator tree, all remaining branches in the separator tree are assigned to the associated leaves of the processor tree. In the event that the separator tree has insufficient depth to cover the processor tree, processors at the leaves would draw upon elements belonging to separators above them in the tree.



(a)

0	0	0	0	2	2	2
0	0	0	0	2	2	2
0	0	0	0	2	2	2
1	1	1	0	2	2	2
1	1	1	3	3	3	3
1	1	1	3	3	3	3
1	1	1	3	3	3	3

(b)



(c)

(d)

Figure 4: (a) Tree-embedding of separator-to-processor distribution for degrees-of-freedom from Fig. 2b. (b) processor mapping for degrees-of-freedom. (c) fan-in communication required during computation of $\underline{c} = X^T \underline{b}$. (d) sparsity pattern and processor distribution for X.

Standard domain decomposition strategies based upon recursive bisection will ensure an appropriate element-to-processor map provided that separator elements are drawn from the partition boundaries generated at each bisection step.

Given a proper ordering and processor distribution, the communication for the fan-in phase (8) of the $XX^T\underline{b}$ evaluation will have the structure illustrated in Fig. 4c. The arrows indicate message sources and destinations for each of the $\log_2 P$ phases as well as the amount of data transmitted. Processors which sum incoming data are denoted in the ovals at each level of the tree. Since the evaluation of each element of $\underline{c} = X^T\underline{b}$ depends only upon descendants of c_j, the amount of information which must propagate up the tree steadily decreases as the summation progress towards the root. For example, the computation of c_j, $\forall j \in S_0$ requires only $|S_0| = \sqrt{n}$ elements to be propagated in the final phase of the fan-in.

The required communication is readily incorporated into the Vector-Sum procedure of Section 2 if the data is sorted according to the global ordering such that $\mu(i,p)$ is monotonically increasing with i for a given processor, p. Suppose that the sequence $s_l^{(p)}$, $l = 0, \ldots, D$, represents the cardinality of the separator sets encountered as one traverses from leaf p of the processor tree to the root (Fig. 4a) and that $m_p := \sum_{l=1}^{D} s_l^{(p)}$. Then the modified Vector-Sum procedure is given by:

Procedure Vector-Sum 2

Gather via binary fan-in

$m_* := 1$
do $l = 1$ to D
  if $mod\,(p, 2^l) = 0$ then
    recv $w(m_*\!:\!m_p)$ from $p + 2^{l-1}$
    $v(m_*\!:\!m_p) := v(m_*\!:\!m_p) + w(m_*\!:\!m_p)$
    $m_* := m_* + s_l^{(p)}$
  elseif $mod\,(p, 2^{l-1}) = 0$ then
    send $v(m_*\!:\!m_p)$ to $p - 2^{l-1}$
    endif
  enddo

Broadcast via binary fan-out

do $l = D$ to 1 by -1
  if $mod\,(p, 2^l) = 0$ then
    $m_* := m_* - s_l^{(p)}$
    send $v(m_*\!:\!m_p)$ to $p + 2^{l-1}$
  elseif $mod\,(p, 2^{l-1}) = 0$ then
    recv $v(m_*\!:\!m_p)$ from $p - 2^{l-1}$
  endif
enddo

At the end of the procedure, each processor has precisely the coefficients required for the final phase of the coarse grid solve, $\underline{x} = X\underline{c}$. Since the amount of data transmitted at each stage is bounded by the number of ancestors for any given leaf, that is, by the number of nonzeros in any row, we conclude that the total communication complexity for the XX^T algorithm is bounded by $2\log_2 P(\alpha + 3\sqrt{n}\beta)$ for the $\sqrt{n} \times \sqrt{n}$ grid problem.

We comment that we explicitly used a depth-first traversal of the tree (Fig. 2d) in developing the separator-based ordering of the degrees-of-freedom. Clearly, the same communication complexity is also obtained if one uses a breadth-first traversal. However, the depth-first traversal guarantees that the nonzero pattern within each column of X is contiguous within each processor, as illustrated in Fig. 4d. Consequently, unit-stride direct addressing can be used during the local dot product (7) and DAXPY (9) phases of the $XX^T\underline{b}$ computation, resulting in improved vectorization and cache performance as well as

reduced memory overhead.

Finally, we note that the work required to generate X via the Gram-Schmidt procedure (11) is $O(n^2)$ and the time is $O(n^2/P)$. These estimates are derived as follows. Let $W(n)$ be the number of operations required to compute X for a $\sqrt{n} \times \sqrt{n}$ grid. The work required to compute the last $\sqrt{n}$ columns of X is essentially the same as the work required to effect $\sqrt{n}$ projections onto X, that is, $2 \cdot 3n^{\frac{3}{2}}$ operations for each of the dot product and DAXPY phases, yielding an operation count of $12n^{\frac{3}{2}}$ per column. Generation of the $\sqrt{n}/2$ basis vectors associated with each of the two second-level separators (S_1 and S_2) involves projections onto matrices with columns of length $n/2$ and at most $2\sqrt{n}$ nonzeros per row, yielding an operation count bounded by $4n^2$. Generation of the vectors associated with all remaining separators comprises four subproblems, each of size $n/4$. Therefore, the total work satisfies the recursion

$$W(n) \;=\; 12\,n^2 \;+\; 4\,n^2 \;+\; 4\,W(\frac{n}{4}) \;\leq\; \frac{4}{3}16n^2.$$

The P-processor time estimate exploits the fact that each of the four subproblems can be treated independently on processor subsets of size $P/4$. Thus,

$$\begin{aligned} T(n,P) &= 12\,\frac{n^2}{P} \;+\; 4\,\frac{n^2}{P} \;+\; \frac{4}{4}\,T(\frac{n}{4},\frac{P}{4}) \\ &= 16\,\frac{n^2}{P} \;+\; 16\,\frac{n^2}{4^2}\frac{4}{P} \;+\; T(\frac{n}{16},\frac{P}{16}) \\ &= 16\,\frac{n^2}{P}\left(1 \;+\; \frac{1}{4} \;+\; \frac{1}{16} \;+\; \ldots\right) \\ &\leq \frac{4}{3}16\,\frac{n^2}{P}\,, \end{aligned} \tag{14}$$

and we conclude that, properly implemented, the Gram-Schmidt procedure attains full P-fold concurrency.

5 Numerical Results

We present measured solution times for the XX^T method on $q \times q$ finite difference meshes for $q = 3, 7, 15, \ldots, 511$. The $n \times n$ matrix A (with $n = q^2$) is derived from a 5-pt stencil discretization of the Poisson problem with Dirichlet boundary conditions. For comparison we provide corresponding times for both the redundant LU and distributed A^{-1} methods. In the table below, the symbol g indicates a granularity restriction (i.e. $n < P$) while m indicates a memory restriction.

Table 1 presents times on the 512 node Intel Paragon XPS at Caltech (50 MHz Intel i860 processors running Paragon OSF/1, release 1.0.4). Note that for $P = 1$ or 2, the redundant LU approach is the fastest method in all cases except for the smallest, where the amount of work is insufficient to allow reliable timings. The redundant LU and A^{-1} approach both suffer memory constraints at values of n which are much smaller than achievable with the XX^T approach. For $n \geq 15^2$, the XX^T approach is the fastest of the three; it is an order-of-magnitude faster than the A^{-1} approach for $n > 127^2$, which is in turn, an order-of-magnitude faster than the redundant LU approach.

The table verifies the assertion made in Section 3 that the A^{-1} approach will be superior to the redundant LU approach whenever $P > n/2s$, where s is the matrix bandwidth for the LU scheme. For this problem, $s = \sqrt{n} = q$, implying that the A^{-1} approach should be superior whenever $P > q/2$. The performance transition is observed at precisely this point for all of the entries in the left half of the table. For the larger problems on the right, the transition occurs at lower values of P as the A^{-1} approach benefits from enhanced vector performance as discussed below.

A careful examination of the operation counts for $n = 63^2$ reveals that the XX^T method should be ten times faster than A^{-1} instead of the observed improvement, which is only four-fold. We found that this is due to the use of the BLAS library DDOT routine on the Paragon which, as is seen in Fig. 5c, shows a sudden 2.25× performance gain for vector lengths greater than ≈ 2330. Since the distributed A^{-1} approach requires DDOTs of length n, whereas the XX^T approach requires DDOTs and DAXPYs of at most length n/P or $3\sqrt{n}$, the former method benefits from this vector performance gain, whereas the latter does not for the values of n considered here.

We note that for the n=31^2, 63^2, and 127^2 cases the efficiency of all of the methods begins to deteriorate as P approaches 512, reflecting that *the solution cost is communication dominated*. These trends are clearly revealed in the plots of solution time vs. number of processors shown in Figs. 5a and b. One would of course expect such trends for the fixed-problem-size speed-up model; the work scales as $1/P$, while the communication scales as $\log_2 P$. What is surprising is the amount of communication overhead suffered by the A^{-1} approach due to bandwidth constraints. The upward swing at the tails of the curves in Fig. 5a reveal the dominance of communication cost, but the magnitude is well above the latency bound, $2\alpha t_a \log_2 P$, which is also plotted. By contrast, the tails of the XX^T curves (Fig. 5b) are much closer to the latency curve, although not as close as might be expected, particularly for the $n = 31^2$ case.

A plot of the total communication overhead for $n \approx P$, added as a dashed line in Fig. 5b, reveals that the XX^T communication costs grow faster than $\log_2 P$ between $P = 256$ and 512. This is explained by a design feature of the Paragon operating system which provides greater bandwidth for smaller numbers of processors, as indicated by the plots of communication time vs. message length shown in Fig. 5d. These times were measured using a standard ping-pong test with non-cached data on successive transfers and asynchronous (i.e., pre-posted) receives for $P = 2, 4, 8, \ldots, 512$. While there is virtually no change in latency as the number of processors increases, there is a five-fold reduction in bandwidth as one moves from $P = 2$ to $P = 512$. This accounts for the faster than $\log_2 P$ growth in communication costs observed in Fig. 5b. Presumably this loss of bandwidth is a result of requiring the system message buffer space to be more finely partitioned in the large P cases. However, in the timings, use of asynchronous receives should have implied that the message buffer memory was managed by the driving application.

Table 1: Paragon solution time in seconds for a $q \times q$ grid						
P	Red. LU	Dist. A^{-1}	XX^T	Red. LU	Dist. A^{-1}	XX^T
	$n = 3\times3$			$n = 63\times63$		
1	4.4600E-05	3.6105E-05	3.4902E-05	1.1949E-01	–	3.4203E-01
2	1.7068E-04	1.3949E-04	1.3573E-04	1.2003E-01	–	1.9369E-01
4	2.9398E-04	2.5764E-04	2.5064E-04	1.2045E-01	m	8.4266E-02
8	4.6827E-04	3.7863E-04	3.5714E-04	1.2107E-01	1.5573E-01	4.1084E-02
16	g	g	g	1.2186E-01	8.0204E-02	2.0343E-02
32	–	–	–	1.2263E-01	4.2857E-02	1.0608E-02
64	–	–	–	1.2431E-01	2.5133E-02	6.2606E-03
128	–	–	–	1.2692E-01	1.7932E-02	4.1330E-03
256	–	–	–	1.3228E-01	1.7967E-02	3.8113E-03
512	–	–	–	1.4916E-01	2.8438E-02	5.0652E-03
	$n = 7\times7$			$n = 127\times127$		
1	2.9800E-04	3.2989E-04	2.7725E-04	9.1016E-01	–	–
2	4.4168E-04	2.7822E-04	2.9301E-04	9.1129E-01	–	–
4	5.8261E-04	3.4939E-04	3.5085E-04	9.1280E-01	–	m
8	7.4149E-04	4.3863E-04	4.3446E-04	9.1395E-01	–	3.5016E-01
16	9.3307E-04	5.6306E-04	5.7672E-04	9.1594E-01	–	1.6388E-01
32	1.1162E-03	6.9726E-04	7.2160E-04	9.1807E-01	–	8.1527E-02
64	g	g	g	9.1976E-01	m	4.1622E-02
128	–	–	–	9.2159E-01	1.7435E-01	2.2244E-02
256	–	–	–	9.2980E-01	1.2968E-01	1.3643E-02
512	–	–	–	1.0379E+00	1.5087E-01	1.1458E-02
	$n = 15\times15$			$n = 255\times255$		
1	2.0880E-03	9.5215E-03	4.5643E-03	–	–	–
2	2.2781E-03	4.9863E-03	2.3550E-03	–	–	–
4	2.4351E-03	2.7819E-03	1.1259E-03	–	–	–
8	2.6284E-03	1.6877E-03	8.2464E-04	–	–	–
16	2.8119E-03	1.1831E-03	7.9471E-04	–	–	–
32	3.1127E-03	1.0036E-03	9.1089E-04	–	–	m
64	3.2951E-03	1.0952E-03	1.0084E-03	–	–	3.2321E-01
128	3.5418E-03	1.1286E-03	1.1770E-03	–	–	1.6368E-01
256	g	g	g	–	–	8.5836E-02
512	–	–	–	m	m	5.3390E-02
	$n = 31\times31$			$n = 511\times511$		
1	1.6356E-02	1.6719E-01	4.5315E-02	–	–	–
2	1.6640E-02	8.3992E-02	2.1006E-02	–	–	–
4	1.6858E-02	4.2522E-02	9.8550E-03	–	–	–
8	1.7122E-02	2.1893E-02	5.1827E-03	–	–	–
16	1.7501E-02	1.1735E-02	3.0003E-03	–	–	–
32	1.7995E-02	6.8829E-03	2.0195E-03	–	–	–
64	1.8841E-02	4.9585E-03	1.6644E-03	–	–	–
128	1.9221E-02	4.0317E-03	1.5370E-03	–	–	–
256	2.0097E-02	4.0358E-03	1.6672E-03	–	–	m
512	2.1847E-02	5.6248E-03	2.3776E-03	m	m	3.6714E-01

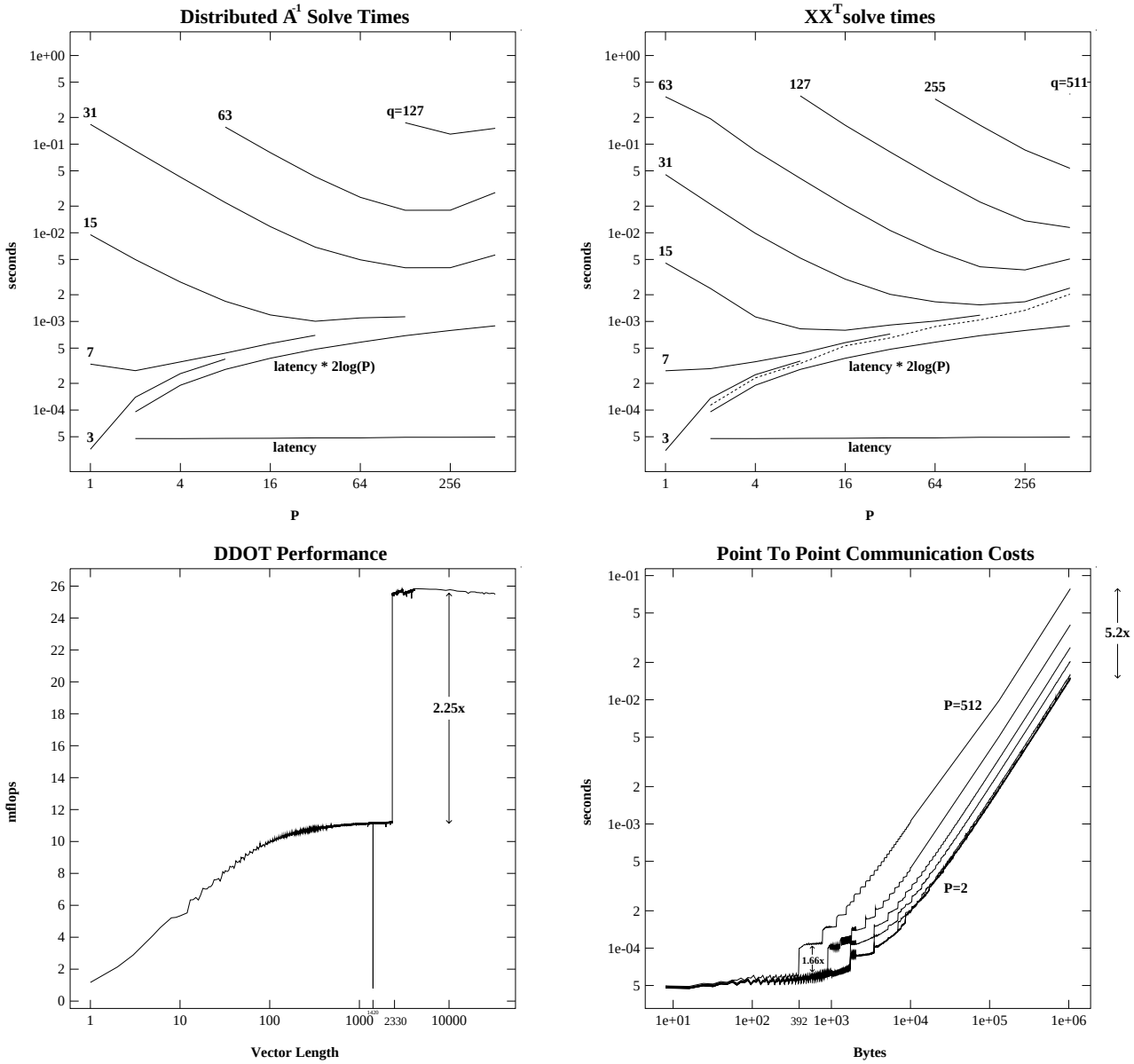


Figure 5: (a) solution times for A^{-1} approach (b) solution times for XX^T approach (c) Paragon DDOT performance *vs.* vector length (d) Paragon communication time *vs.* message size.

Figure5d also reveals that the linear communication model (3) is adequate at the small and large message limits but does not capture sudden transitions in communication cost which may be significant in actual measured applications. As these nonlinear features are hardware and operating system dependent there is little one can do to incorporate them into generic complexity estimates in any meaningful way.

Finally, in Fig. 6 we present results for $n = 63^2$ and 127^2 on the Intel ASCI ASCI-Red machine at Sandia National Laboratories (333 MHz Pentium IIs with Xeon Core Technology running TOS/Cougar OS R3.0). In comparison to the Paragon, ASCI-Red has relatively high latency but also high bandwidth, representative figures being $(t_a, \alpha t_a, \beta t_a) \approx (.1\mu s, 50\mu s, 0.68\mu s/\text{64-bit-word})$ for the Paragon, and $\approx (.01\mu s, 20\mu s, .02\mu s/\text{64-bit-word})$ for ASCI-Red. We can therefore expect to find ASCI-Red solution times closer to the

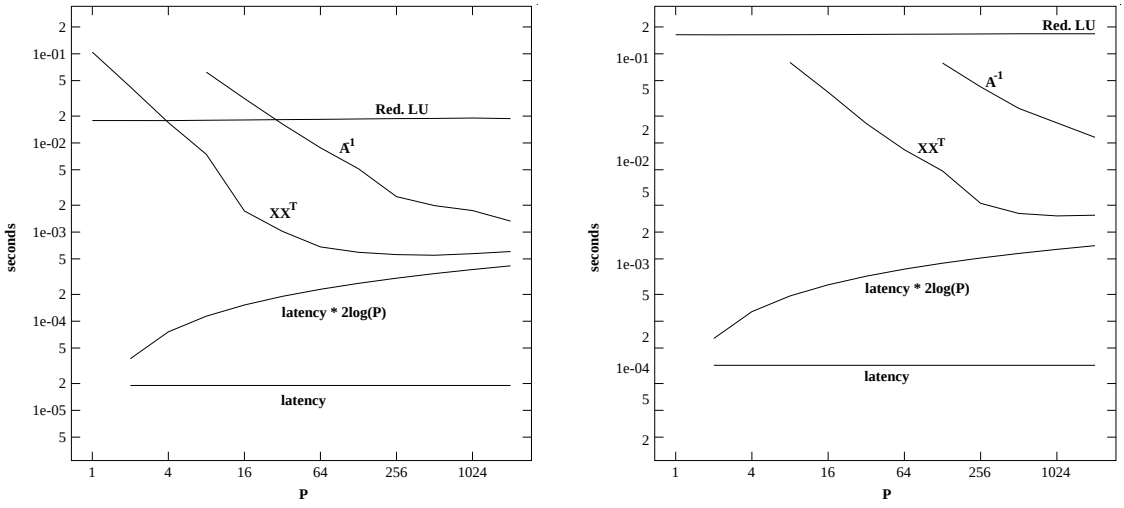


Figure 6: ASCI-Red-333 solve times for $n = 63^2$ (left) and 127^2 (right).

$2\alpha t_a \log_2 P$ bound in the communication dominated regime ($P \approx n$) since the bandwidth term will contribute less. In fact, the A^{-1} and XX^T approaches should converge in the limit of infinite bandwidth since both schemes have the same latency costs. Evidence of this can be seen for the $n = 63^2$, $P = 2048$ case in Fig. 6.

It is important to note that the redundant LU approach could potentially be improved for the larger problems by using a nested dissection ordering. From [11], the number of nonzeros in the triangular factors for a nested-dissection ordering is only

$$nnz \;=\; \frac{93}{12}(q+1)^2\log_2(q+1) \;-\; \frac{73}{3}(q+1)^2 \;+\; 24(q+1)\log_2(q+1) \;+\; O(q),$$

in contrast to $O(q^3)$ for the banded factorizations used in the above timings. Optimistically, we can estimate the work savings as the ratio of the number of nonzeros, which corresponds to a factor roughly 2.5 for $q = 63$ and 4.1 for $q = 127$. Realistically, the savings will be less dramatic due to the indirect addressing overhead and loss of vectorization and we can conclude that the BLAS-based performance figures presented here are fairly representative of the redundant solve performance.

6 Discussion

We can use the complexity estimates of the previous sections to estimate the cost of the XX^T algorithm on state-of-the-art supercomputers using thousands of processors, such as are currently under development at the national laboratories. We take as an example the two-dimensional model problem on $P = 8192 = 2^{13}$ processors, with $n = P$ (the most challenging case), and use the communication constants measured on ASCI-Red. To estimate the computational cost, we assume a conservative value of $t_a = 5.0 \times 10^{-8}$ ops/second, corresponding to 20 MFLOPS. The communication costs for $XX^T\underline{b}$ are thus

$$\begin{aligned} T_c &= 2\alpha t_a \log_2 P \;+\; 3\sqrt{n}(2\beta+1)t_a \log_2 P \\ &= 5.2\times 10^{-4} \;+\; 2.1\times 10^{-4}\text{seconds}\,, \end{aligned}$$

and the computational cost is

$$\begin{aligned} T_a &= (4 \cdot 3n\sqrt{n}/P)\, t_a \\ &= 0.54 \times 10^{-4} \text{seconds}\,. \end{aligned}$$

These results reveal that, under reasonable model assumptions, the bandwidth cost (0.21 msec) is half the latency cost (0.52 msec), and that the arithmetic cost (0.054 msec) is a order-of-magnitude smaller than the bandwidth cost. Because of the significance of the latency term, it is clear that competing methods will have to adhere to the strategy of using a minimum number of messages. Moreover, as the bandwidth cost dominates the arithmetic cost, it is more important to focus upon reductions in message traffic than further reductions in work.

It is possible that the number of nonzeros in X (and hence, communication) can be reduced by carefully selecting the generating basis, V, to yield a number of entries in X that are below a given threshold value and which could therefore be neglected, resulting in an acceptable approximation to A^{-1}. This is particularly true in the case where A is being used as part of a preconditioner, in which case the exact solution to (1) is not required. Related ideas in the area of approximate inverse factorizations based on thresholding have been explored by a number of authors (see, e.g., [2, 3, 16]). One promising approach in this regard is to recognize that the computational complexity bounds derived in Section 4 are independent of the ordering of basis vectors within a given separator S_l. Therefore, rather than using successive unit vectors, $\underline{\hat{e}}_i$, $i \in S_l$, one might choose a Fourier-like basis having the form,

$$\begin{aligned} \underline{v}_i &= \{...001111111100...\}^T \\ \underline{v}_{i+1} &= \{...001111000000...\}^T \\ \vdots &\qquad\qquad \vdots \\ \underline{v}_{i-1+|S_l|} &= \{...00\underbrace{1010101010}_{\text{Support of } S_l}00...\}^T\,, \end{aligned} \tag{15}$$

and which vanish outside the support of S_l. Applied to each separator, this leads to a fill pattern more closely resembling that shown in Fig. 1d rather than a strictly upper-triangular factor. Because the basis vectors $\underline{x}_j$ decay smoothly away from the separators (at least for elliptic problems), this highly oscillatory generating basis should yield columns in X which are effectively zero away from the separator. Initial results for Poisson's equation on a square have shown that the bases (15) do indeed lead to a greater number of "small" entries in X, and to better round-off properties. However, it appears that a smoother set of oscillatory basis functions will ultimately be required if significant savings are to be realized from such a thresholding strategy.

Another common strategy for improved performance is to solve the coarse grid problem cooperatively (and redundantly) among processor subsets. The cooperative solve can be implemented with any of the approaches discussed previously, including the XX^T approach. Of course, this does not circumvent the $\log_2 P$ bound on the minimum number of messages and so does little to reduce latency. However, it could be used to ameliorate nontrivial

bandwidth limitations. Our suggested strategy is to gather segments of the right-hand side, $\underline{b}$, onto independent processor subsets using l' rounds of the recursive doubling variant of the Vector Concatenate routine of Section 2. Here, l' is determined such that the message size in the l'th round of the concatenation is equal to a threshold value, m'. For example, let $m' = \min(\alpha/\beta, 3\sqrt{n})$ and choose l' such that $2^{l'-1}n/P = m'$. This ensures that the recursive doubling variant of concatenation does not suffer line contention (since messages shorter than α/β are latency dominated) and does not require message lengths exceeding those required by the XX^T algorithm. After l' rounds of recursive doubling, the coarse grid problem can be solved with the XX^T algorithm using only $2(\log_2 P - l')$ rounds of communication. A similar strategy is employed when $P \neq 2^D$. One identifies the largest value of D such that $P' = 2^D < P$, and maps the right-hand side data from processors $p = P', \ldots, P-1$, onto respective counterparts in $\{0, \ldots, P'-1\}$. The solution is then computed using $P' = 2^D$ processors following the strategy outlined in Section 4.

The XX^T method can be extended to non-symmetric problems by altering the projection to minimize in the A^TA-norm, as is standard for projection-based iterative methods such as GMRES. The basis for the procedure in this case is initiated by seeking unit vectors satisfying $(A\underline{\hat{e}}_i)^T A\underline{\hat{e}}_j = 0$, that is, that are A^TA-conjugate. From considerations similar to those presented in Fig. 1, it is clear that this is achieved by simply choosing separators of width two rather than unity. All of the computational and communication complexity bounds follow immediately in $\mathbb{R}^2$ or $\mathbb{R}^3$, and the nonsymmetric solver will require twice the storage, twice the amount of data traffic, and precisely the *same* number of messages ($2\log_2 P$) as its symmetric counterpart. For a small increment in storage, one can avoid the need to multiply by A^T by storing columns $\underline{y}_j := A\underline{x}_j$ such that the solution is computed as $\underline{x} = XY^T\underline{b}$. In this case, XY^T does not correspond to a triangular factorization of A^{-1}.

Finally, because of the generality of the graph-partitioning arguments and the binary tree embeddings employed, the XX^T method readily extends to fully general mesh problems. For complex two- or three-dimensional meshes, separator sets can be found with standard graph-splitting techniques (e.g. recursive coordinate bisection) or via one of the many variants of recursive spectral bisection (e.g., [17]). In general, one can expect somewhat smaller complexity constants than for the examples considered here, as $\sqrt{n}$ is generally the worst-case separator bound for planar graphs. Provided that subdomains are mapped according to the separator induced partitioning, the general geometry implementation will enjoy the same low communication requirements as the very regular examples considered here.

We have used the algorithm in numerous parallel spectral element solutions of the incompressible Navier-Stokes equations in two- and three-dimensional domains [8, 19], including the example illustrated in Fig. 7. In these applications, the coarse grid operator derives from a linear finite element discretization of Poisson's equation on the spectral element vertices and is used in conjunction with an overlapping Schwarz procedure to accelerate the convergence of the conjugate-gradient-based pressure computation at each time step. Recursive spectral bisection is used to distribute the elements to processors. Because the spectral element data is partitioned by elements, vertex data for the coarse grid problem is stored redundantly, implying that extra communication is nominally required to copy

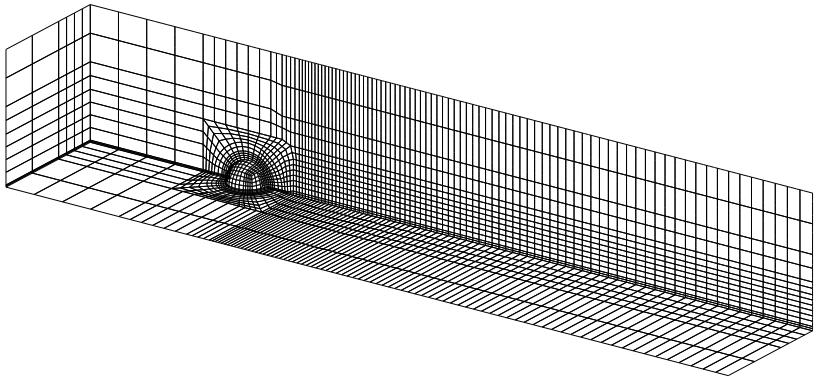


Figure 7: Spectral element mesh used to simulate the interaction of a flat-plate boundary layer flow with a hemispherical roughness element.

the computed entries of $\underline{x}$ to multiple recipients. By symmetry, it follows that a gather is required in constructing $\underline{b}$. It is possible to avoid the extra communication by embedding these steps directly into the XX^T procedure. If Q represents the copy operation, then the full solution procedure is $\tilde{\underline{x}} = QXX^TQ^T\tilde{\underline{b}} = \tilde{X}\tilde{X}^T\tilde{\underline{b}}$, where $\tilde{X} = QX \in \mathbb{R}^{m\times n}$, $m > n$, represents the original basis of X with each element replicated onto any processor sharing its corresponding vertex. This results in an increase in the amount of work (at most $8\times$) but does *not* increase either the number of messages or the size of the messages required to effect a solve. Further details can be found in [8].

The three-dimensional mesh problem in Fig. 7 is from [19] and contains $E = 8168$ elements of polynomial order $N = 15$ (27,799,110 gridpoints for velocity, 22,412,992 for pressure). The number of spectral element vertices is $n = 10142$. On 2048 nodes of ASCI-Red, the number of words communicated in each of the $\log_2 P$ stages of the contention-free fan-in is given in Table 2. (See Fig. 4c.) It is clear that these values are well below the $O(n)$ costs incurred by the methods discussed in Section 3, and in fact are quite close to the $\approx 2.33n^{\frac{2}{3}}$ complexity bound one would expect for a regular three-dimensional mesh. Performance results for the first 26 time steps of the calculation are given in Table 3. As the XX^T solver accounts for less the one percent of the Navier-Stokes solution time, it is possible to now consider a richer coarse grid space to enhance convergence rates.

We close with a comparison of the $P = 2048$ result to theory. We use αt_a and βt_a from the two-dimensional model problem prediction but increase t_a to 2.0×10^{-8} ops/second, corresponding to 50 MFLOPS, to reflect the (measured) increase in performance due to longer

Table 2: # words sent at each stage of binary fan-in

stage	10	9	8	7	6	5	4	3	2	1	0
min	546	546	546	543	534	516	483	425	327	223	119
max	780	772	766	753	739	721	680	610	481	271	119

Table 3: XX^T Performance on ASCI-Red					
	$K = 8168$, $N = 15$				
P	NS time (s)	NS GFLOPS	XX^T %	calls	time per call (s)
256	9025	33	0.2	4863	0.0033
512	4537	65	0.3	4923	0.0032
1024	2242	132	0.6	4841	0.0027
2048	1148	257	0.9	4484	0.0022

DDOTs and DAXPYs. The communication estimate is

$$\begin{aligned} T_c &= 2\alpha t_a \log_2 P \; + \; \frac{7}{3} n^{\frac{2}{3}} (2\beta + 1) t_a \log_2 P \\ &= 4.4 \times 10^{-4} \; + \; 7.2 \times 10^{-4} \text{seconds}\,, \end{aligned}$$

and computational estimate is

$$\begin{aligned} T_a &= (4 \cdot \frac{7}{3} n^{\frac{2}{3}} \cdot n/P)\, t_a \\ &= 4.3 \times 10^{-4} \text{seconds}\,. \end{aligned}$$

Predicted performance is thus 1.6 msec while actual performance is 2.2 msec. The discrepancy is accounted for by the fact that the total amount of work is increased by approximately a factor of two due to the embedding procedure.

We conclude that the relatively low computational complexity and excellent communication complexity of the XX^T-based solver will make it a very competitive algorithm for leading edge multicomputer systems. Moreover, as the coarse grid solve is central to efficient iterative solution of many systems arising from partial differential equations, fast coarse grid solvers such as presented here will be critical to future Teraflops applications.

Acknowledgments

This work was supported by the NSF under Grant ASC-9405403 and by the AFOSR under Grant F49620-95-1-0074; by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38; by the Department of Energy under Grant No. B341495 to the Center on Astrophysical Thermonuclear Flashes at University of Chicago, and by the University of Chicago. Computer time was provided on the Intel Paragon and Delta at Caltech under NSF Cooperative agreement CCR-8809615 and on the Intel Paragon at Wright Patterson Air Force Base by the AFOSR.

References

[1] F. Alvarado, A. Pothen, and R. Schreiber, "Highly parallel sparse triangular solution", Univ. Waterloo Research Rep. CS-92-51, Waterloo, Ontario (1992).

[2] M. Benzi, J. Marin, and M. Tuma, "A two-level parallel preconditioner based on sparse approximate inverses", in *Iterative Methods in Scientific Computation IV*, D. R. Kincaid et. al. eds., IMACS, pp. 167–177 (1999).

[3] M. Benzi, C. D. Meyer, and M. Tuma, "A sparse approximate inverse preconditioner for the conjugate gradient method", *SIAM J. Sci. Comput.*, **17** 5, pp. 1135–1149 (1995).

[4] X.-C. Cai, "The use of pointwise interpolation in domain decomposition with non-nested meshes," *SIAM J. Sci. Comput.*, **14** 1, pp. 250–256 (1995).

[5] T. Chan and J.P. Shao, "Parallel complexity of domain decomposition methods and optimal coarse grid size", *Parallel Computing*, **21**, pp. 1033–1049 (1995).

[6] M. Dryja and O. Widlund, "Towards a unified theory of domain decomposition algorithms for elliptic problems", in *Proceedings of the Third International Symposium on Domain Decomposition Methods for Partial Differential Equations*, R. Glowinski, J. Periaux, and O. Widlund, eds., SIAM, Philadelphia, pp. 3–18 (1990).

[7] C. Farhat and P.S. Chen, "Tailoring domain decomposition methods for efficient parallel coarse grid solution and for systems with many right hand sides", *Contemporary Math.*, **180**, pp. 401–406 (1994).

[8] P. F. Fischer, N. I. Miller and H. M. Tufo, "An overlapping Schwarz method for spectral element simulation of three-dimensional incompressible flows", in *Parallel Solution of Partial Differential Equations*, P. Bjørstad and M. Luskin, eds., Springer-Verlag pp. 159–181 (2000).

[9] P. F. Fischer, "Parallel multi-level solvers for spectral element methods", in *Proceedings of Intl. Conf. on Spectral and High-Order Methods '95, Houston, TX*, Houston J. of Math., R. Scott, Ed., pp. 595–604 (1996).

[10] P. F. Fischer, "Parallel domain decomposition for incompressible fluid dynamics", *Contemporary Mathematics*, **157**, pp. 313–322 (1994).

[11] J.A. George, "Nested dissection of a regular finite element mesh", SIAM J. Numer. Anal., **15**, pp. 1053–1069 (1978).

[12] W.D. Gropp, "Parallel computing and domain decomposition", In *Fifth Conference on Domain Decomposition Methods for Partial Differential Equations*, T.F. Chan, D.E. Keyes, G.A. Meurant, J.S. Scroggs, and R.G. Voigt, eds., SIAM, Philadelphia, PA, 1992.

[13] W.D. Gropp and D.E. Keyes, "Domain decomposition with local mesh refinement", SIAM J. Sci. Stat. Comput., **15** 4, pp. 967–993 (1992).

[14] D.E. Keyes, Y. Saad, and D.G. Truhlar, *Domain-Based Parallelism and Problem Decomposition Methods in Computational Science and Engineering*, SIAM, 1995.

[15] J.L. Gustafson, G.R. Montry, and R.E. Benner, "Development of Parallel Methods for a 1024-Processor Hypercube", *SIAM J. on Sci. and Stat. Comput.*, **9** 4, pp. 609–621 (1988).

[16] L. Yu. Kolotilina and A. Yu. Yeremin, "Factorized sparse approximate Inverse Preconditionings I. Theory", *SIAM J. Matrix Anal. Appl.*, **14** 1, pp. 45–58 (1993).

[17] A. Pothen, H.D. Simon, and K.P. Liou, "Partitioning sparse matrices with eigenvectors of graphs", *SIAM J. Matrix Anal. Appl.*, **11** 3, pp. 430–452 (1990).

[18] B. Smith, P. Bjørstad, and W. Gropp, *Domain Decomposition*, Cambridge University Press, 1996.

[19] H.M. Tufo and P. F. Fischer, "Terascale spectral element algorithms and implementations", SC99, http://www.sc99.org/proceedings/papers/tufobell.pdf (1999).

[20] R. van de Geijn, "On global combine operations", *J. Parallel & Distributed Comput.*, **22**, pp. 324–328 (1994).

[21] O.B. Widlund, "Iterative substructuring methods: algorithms and theory for elliptic problems in the plane", in *Proceedings of the First International Symposium on Domain Decomposition Methods for Partial Differential Equations*, R. Glowinski, G.H. Golub, G.A. Meurant, and J. Periaux, eds., SIAM, Philadelphia, (1988).