Fast Parallel Direct Solvers For Coarse Grid Problems

H. M. Tufo
P. F. Fischer *

May 23, 1997

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 the dimension of the system is n and the number of processors is P, our approach requires $O(n^{\gamma} \log_2 P)$ time for communication and $O(n^{1+\gamma}/P)$ time for computation, where $\gamma \equiv \frac{d-1}{d}$. Results from a 512 node Intel Paragon show that our algorithm compares favorably to more commonly used approaches which require $O(n \log_2 P)$ time for communication and $O(n^{1+\gamma})$ or $O(n^2/P)$ time for computation. Moreover, for leading edge multicomputer systems with thousands of processors and $n = P$ (i.e., communication dominated solves), we expect our algorithm to be markedly superior as it achieves substantially reduced message volume and arithmetic complexity over current competing methods while retaining minimum message startup cost.

Presented at the 13th annual conference of the *Intel Supercomputer Users Group*, Sandia National Laboratory, Albuquerque, New Mexico, June 13, 1997.

*Division of Applied Mathematics, Brown University, Providence, RI 02912, USA. E-mail: *pff@cfm.brown.edu*, *hmt@cfm.brown.edu*

1 Introduction

In this paper we consider 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 matrices arise when discretizing d-dimensional elliptic PDE's by standard low-order finite differences or finite elements. We target our algorithm for the "fine-grained" regime (i.e., $n/P \approx 1$) and large P. Such fine-grained solvers are required for the solution of the coarse grid systems encountered when using multigrid or domain decomposition methods (e.g.[3, 4]). This work focuses on the *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. Per force, the 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.

The problem of the coarse-grid solve has been studied widely in the domain decomposition community. It has been established by Widlund [4] that order-independent convergence rates in domain decomposition methods cannot be obtained without the solution of a coarse grid problem. In [3], 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.* [9, 11, 13] also discuss the importance and challenge of developing an efficient parallel coarse grid solver for domain decomposition methods. Cai [2] 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 their scheme for the coarse grid problem must require communication in the prolongation/restriction phase or have a minimum of one degree-of-freedom per processor.

Since leading edge multicomputer systems are currently scaling to thousands of processors, there is a clear need for an efficient treatment of the coarse grid problem. In particular, if the work per processor remains constant while the number of processors increases (the standard model for scaled speed up [12]), the coarse grid problem will ultimately dominate the complexity unless the solve time can be substantially reduced.

In this paper we discuss a fast parallel coarse grid solution algorithm based upon creating a sparse A-conjugate basis,

$$X = (\underline{x}_1, \ldots, \underline{x}_n)\,, \tag{2}$$

for $\mathbb{R}^n$. It can be shown that this constitutes a sparse factorization (not necessarily triangular) of the full matrix A^{-1}. The scheme has a per-solve complexity of $O(n^\gamma \log_2 P)$ 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 $O(n \log_2 P)$ time for communication and $O(n^{1+\gamma})$ or $O(n^2/P)$ time for computation. Results obtained on a 512 node Intel Paragon show that our method performs well even for values of n/P significantly greater than unity, particularly for larger values of P.

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 existing 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 and discuss expected performance on thousands of processors. Finally, current research efforts and possible extensions of the algorithm 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 computation of

$$\underline{x} = A^{-1}\underline{b}, \tag{3}$$

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 handle 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 (3) is $\log_2 P$; we take this as a lower bound on the solution time.

We assume a standard model for contention-free interprocessor communication in which the time to send an m-word message is given by

$$t_c[m] = (\alpha + \beta m)t_a\,, \tag{4}$$

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, $\alpha \gg \beta \gg 1$. Thus, optimal communication strategies may possibly require a balance between minimizing the *number* of messages as well as the *length* (m) of the messages. We 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. The second computes

$$\underline{v} = \sum_p \underline{v}^{(p)}\,, \tag{5}$$

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 $p + 2^{l-1}$
    else
      send $v(1{:}\hat{m})$ to $p - 2^{l-1}$
      goto 1
    endif
    $\hat{m} := 2\hat{m}$
  enddo

Broadcast via binary fan-out

1 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 $p + 2^{l-1}$
      $v(1{:}m) := v(1{:}m) + w(1{:}m)$
    else
      send $v(1{:}m)$ to $p - 2^{l-1}$
      goto 1
    endif
  enddo

Broadcast via binary fan-out

1 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

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, for a total time of approximately $(2\alpha + \beta m)\log_2 P\, t_a$. The vector sum requires a total time of $2(\alpha + \beta m + 1/2)\log_2 P\, t_a$.

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 requires a total time of $(\alpha \log_2 P + \beta \cdot (m - m_p))t_a$, nominally a factor of $\log_2 P$ better than the fan-in/fan-out approach, while the vector sum routine requires a time of $(\alpha + \beta m + 1)\log_2 P\, t_a$, nominally a factor of two superior to the fan-in/fan-out approach. However, on low-dimensional networks the recursive doubling schemes will suffer both node and line contention unless $m \lesssim \alpha/\beta$, and performance will generally be inferior to the fan-in/fan-out approach. For concatenation, hybrid approaches are possible, e.g., in which recursive doubling is used until $\hat{m} \approx \alpha/\beta$, and fan-in/fan-out is used on processor subsets to span the remaining levels of the tree(s). In addition, for $m \gg \alpha/\beta$, there is a well known hybrid scheme for vector summation due to van de Geijn *et al.* [14]. For the values of m considered here, the fan-in/fan-out schemes capture the essential complexity, while hybrid schemes would be most appropriate as a fine tuning measure in the final phases of implementation, so 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 implementation of matrix-vector multiplication on $P = 2^D$ processors. Assume $\underline{x}$ is a vector having n components, $(x_1, x_2, \ldots, x_n)$, which are distributed across P processors according to the bijective map $\mu(i,p) \in \{1, \ldots, n\}$, 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, i.e., $n_p = n/P$. 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* 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 μ'th row of C. The end result is a vector $\underline{y}$ which is distributed according to the mapping 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. We remark that the DAXPY based approach in which the communication is performed after the computation can be implemented with identical complexity.

3 Survey of coarse grid solvers

It is well known (e.g., [9]) that parallel solution of the coarse grid problem is hampered by the inherent sequentiality of the forward and backward substitution phases of standard LU (or LL^T) factorizations. 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 $2\alpha \log_2 P + 2\beta n$. 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 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., [6, 10]). 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 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 costs of n^2/P per processor limit this approach to values of n (and hence, P) of up to only a few thousand. With the advent of computers containing thousands of processors this restriction becomes problematic. Ideally, one would like a matrix-vector product approach involving *sparse* matrices.

A step in this direction is the approach of Alvarado *et al.* [1] who develop fast parallel triangular solvers by recasting the inverse of a 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 $\underline{v}_0 = \underline{b}$, $\underline{v}_i = \tilde{L}_i^{-1}\underline{v}_{i-1}$, $\underline{x} = \underline{v}_l$. 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., [8]). Since solution of (1) requires both a forward- and backward-sweep, we estimate a lower bound on the solution time as $2\alpha(\log_2 P)^2 t_a$. This estimate neglects both the work, which, with a lower bound of $\Omega(\frac{n \log_2 n}{P})$ [8], probably is negligible, and the amount of data communicated, which probably is not negligible.

Another approach of interest is that of Farhat and Chen [5] who solve the coarse grid problem by projecting onto sets of previously generated Krylov vectors which constitute an approximation space. If $X_k = (\underline{x}_1\ \underline{x}_2\ \ldots\ \underline{x}_k)$ is a matrix of k A-conjugate vectors, normalized to satisfy $\underline{x}_i^T A \underline{x}_j = \delta_{ij}$, then

$$\bar{\underline{x}} = X_k X_k^T \underline{b} \tag{6}$$

yields the projection onto $\mathcal{R}(X_k)$. Here, δ_{ij} is the Kronecker delta and $\mathcal{R}(\cdot)$ is the range of the argument. 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. In the examples reported in [5], 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 bound 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, $\underline{x}_i$, 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 begins by computing an intermediate k-vector, $\underline{c} = X^T \underline{b}$, in two stages, beginning with evaluation of the local dot products:

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

followed by a $\log P$ sum across processors

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

With the components of $\underline{c}$ known to every processor, the final stage in computing the distributed vector $\bar{\underline{x}}$ is simply

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

This final stage is recognized as a sequence of k DAXPY's of length n_p on each processor p, and is fully parallel. If the vectors $\underline{x}_i$ are full, this approximation has leading order complexity of $4nk/P$ operations for the required dot-products (7) and DAXPY's (9), and communication time $2 \log_2 P(\alpha + \beta k) t_a$ for the gather of the k coefficients of each column vector (8).

Note that if $k = O(n)$, then the projection approach is at best better than the full A^{-1} approach by only a constant, with a lower bound solution time of $2\alpha(\log_2 P) t_a$ obtained for both methods. In the next section, we present a method in which we take $k \equiv n$ but which, by virtue of choosing a *sparse* basis set, achieves a complexity which is of lower order than the A^{-1} approach. Moreover, this approach is implemented with minimal latency (minimal number of message cycles), and also has a lower bound time of $2\alpha(\log_2 P) t_a$.

4 Sparse basis projection method

The goal of the method of Farhat and Chen [5] 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 $\bar{\underline{x}} \equiv \underline{x}$, implying that $X_n X_n^T$ is the inverse of A. Hence, we can modify the projection approach to by seeking a full set of basis vectors, $X \equiv X_n$, which is as sparse as possible. In [7], we introduced such a method. Here, we describe its implementation on the 512 node Intel Paragon, discuss communication considerations, and performance on a $\sqrt{n} \times \sqrt{n}$ model problem.

We begin with the following construction. Let the unit vectors $\hat{\underline{e}}_i$ and $\hat{\underline{e}}_j$ denote the i'th and j'th column of of the identity matrix, I_n. Let $\mathcal{N}_j$, the *neighborhood* of j, be the set of row indices corresponding to non-zeros in column j of A. Then

$$\hat{\underline{e}}_i^T A \hat{\underline{e}}_j \quad = \quad 0 \qquad \forall\, i \notin \mathcal{N}_j \quad . \tag{10}$$

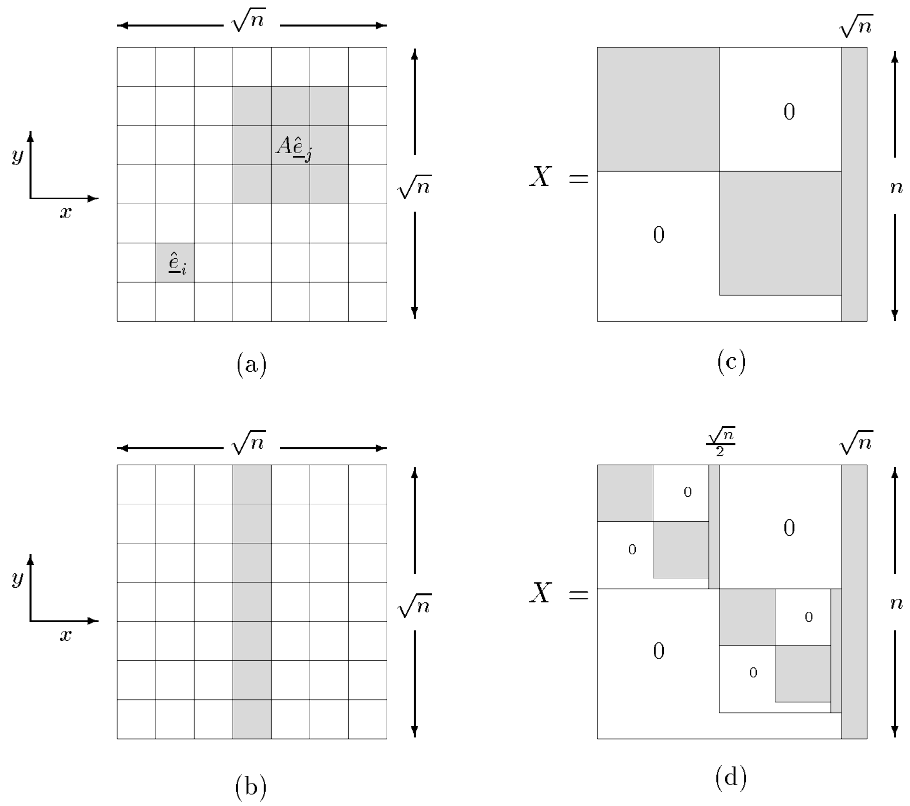
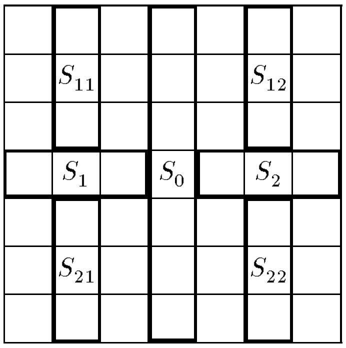


Figure 1: (a) geometric support (shaded) of orthogonal vectors $\underline{\hat{e}}_j$ and $A\underline{\hat{e}}_i$. (b) support of separator set. (c) zero/fill structure for X resulting from ordering the separator set last. (d) zero/fill structure after second round of recursion.

Geometrically, this corresponds to the situation shown in Fig. 1a for a 9-point stencil. It is clear that at least $n/\max(\#\mathcal{N}_j)$ of the unit vectors are A-conjugate to one another, where $\#\mathcal{N}_j$ is 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 by $1/\sqrt{a_{jj}}$. The first such k_1 columns of X will each have *only one non-zero 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 permutation of the identity matrix. Then the procedure

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

ensures that X is the desired factor of A^{-1}. For $k \leq k_1$ the projection $X_{k-1}X_{k-1}^T A\underline{v}_k$ 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. Then goal then is to find an ordering which results is minimum or near minimum fill for the factor X.



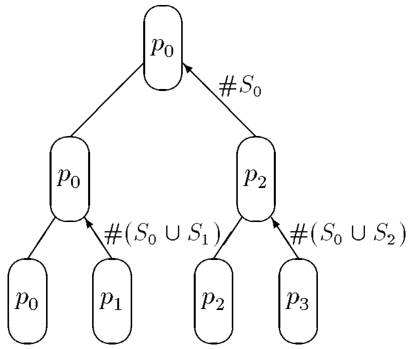
(a)

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

(b)

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

(c)



(d)

Figure 2: (a) separator sets (only first three levels shown). (b) global numbering induced by ordering separator sets last. (c) element to processor map. (d) solver communication tree.

Following [8], we observe that an efficient procedure for selecting such a permutation matrix, V, can be developed by defining separators which recursively divide the domain (or graph) associated with A into nearly equal subdomains. For a $\sqrt{n} \times \sqrt{n}$ grid the first such separator is shown in Fig. 1b. Since the stencil for A does not cross the separator, it is clear that every unit vector $\underline{\hat{e}}_i$ associated with the left half of the domain in Fig. 1b is A-conjugate to every unit vector $\underline{\hat{e}}_j$ 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 reordered according to the permutation used for the columns of V). The 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 (6) that the computational complexity of each solve is proportional to the amount of 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

$$\sqrt{n} + \frac{\sqrt{n}}{2} + \frac{\sqrt{n}}{2} + \frac{\sqrt{n}}{4} + \frac{\sqrt{n}}{4} + \ldots \leq 3\sqrt{n}\,, \tag{11}$$

implying a total bound on the amount of fill in X of $3n\sqrt{n}$. Since we can evenly distribute the work amongst 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-dimension cases provide us with 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. For properly mapped two-dimensional problems, we now show that it is possible to obtain a contention-free communication complexity of $(\alpha \log_2 P + O(n^{\frac{1}{2}})\beta \log_2 P)$, even on a linear array of processors. In Fig. 2 we show three levels of separators for a 7×7 grid (a), the associated permutation which yields the fill pattern of Fig. 2d (b), the associated processor mapping for $P = 4$ (c), and the binary-tree embedding of the communication pattern (d).

We note that columns in X associated with S_0 are nonzero in every row, and that each processor must make a contribution to the corresponding entry in $\underline{c} = X^T\underline{b}$. In addition, for the DAXPY phase of the computation, each processor requires knowledge of those coefficients of $\underline{c}$ which are associated with S_0. Such information is universally required *only* of the elements associated with S_0. For example, Processor 1 makes no contribution to components of $\underline{c}$ associated with elements of S_2, and has no need of the S_2 coefficients in the DAXPY phase. Hence, in the final phase of the binary fan-in, shown in Fig. 2d, only $\sqrt{n} = \#S_0$ data is propagated up the tree. In the second level of the tree, we have $\sqrt{n} + \frac{\sqrt{n}}{2}$ data transmitted in the fan-in and fan-out stages. Summing over all stages gives a total communication bound of $2t_a\,(\alpha + 3\beta\sqrt{n})\log_2 P$. Similarly, for three-dimensional problems, the complexity is $2t_a\,(\alpha + O(n^{\frac{2}{3}})\beta \log_2 P)$.

5 Numerical Results

Solution times for the XX^T method on $q \times q$ (with $n = q^2$) finite difference meshes for $q = 3, 7, 15, \ldots, 511$ are presented in Table 1. The matrix A we used was generated by 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. Note that all times were generated on the 512 node Intel Paragon at Caltech running Paragon OSF/1, release 1.0.4. Also, note that the symbol g in the table indicates a granularity restriction (i.e. $n < P$) while m indicates a memory restriction.

Note that that for $P = 1$ or 2, the redundant LU approach is the fastest method in all cases except for the 3×3 case which is too small to really give 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 $n \geq 225$, the XX^T is the fastest of the three methods; it is an order-of-magnitude faster than the A^{-1} approach for $n > 16129$, which is in turn, an order-of-magnitude faster than the the redundant LU approach.

A careful examination of the operation counts for the A^{-1} approach and XX^T methods reveals that for $n = 3969$, the XX^T solver should be ten times faster than the A^{-1} scheme, instead of the observed improvement which is only four-fold. We found that this is due to the use of the assembly coded DDOT on the Paragon, which has a 2.5 gain in MFLOPS for

Solution time in seconds for a $q \times q$ grid on p processors

p	Red. LU	Dist. A^{-1}	XX^T	Red. LU	Dist. A^{-1}	XX^T
	$n = 3{\times}3$			$n = 63{\times}63$		
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
	$n = 7{\times}7$			$n = 127{\times}127$		
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
	$n = 15{\times}15$			$n = 255{\times}255$		
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
	$n = 31{\times}31$			$n = 511{\times}511$		
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

vector lengths greater than ≈ 2200. Since the distributed A^{-1} approach requires DDOT's of length n, whereas the XX^T approach requires DDOTS's and DAXPY's of at most length $3\sqrt{n}$, the former method benefits from this vector performance gain, whereas the latter does not unless n is significantly increased.

Finally, we note that for the $n = 16129$ case, the efficiency of all of the methods begins to deteriorate as P approaches 512. This is to be expected for this problem class as the work (for the latter two methods, at least) scales as $1/P$, while the communication scales as $\log P$. For $P = 10,000$ we expect that essentially all of the time will be spent on communication. The fact that the XX^T method has a much better constant, and requires a minimal number of messages, implies that it will be a very competitive algorithm in future large scale parallel applications.

6 Extensions

We are currently extending this method to general finite element problems. For more 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 variants of recursive spectral bisection.

In addition, this method can be applied to non-symmetric problems by factoring $(A^TA)^{-1}$. The asymptotic complexity remains unchanged as the operation count is only increased by a factor of two. Since the target application is for coarse grid problems, the ill-conditioning incurred by using A^TA should not be constraining.

Finally, for elliptic problems, the basis functions are very smooth in regions away from the separators. It is therefore possible that further reductions in the number of nonzeros may be attained by carefully selecting the generating bases, V, and thresholding small entries in X. This is particularly true in the case were A is being used as a preconditioner, in which case the exact solution to (1) is not required.

Acknowledgments

This work was supported by the NSF under Grant ASC-9405403 and by the AFOSR under Grant F49620-95-1-0074. Computer time was provided on the Intel Paragon and Delta at Caltech by the Center for Research on Parallel Computation 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] X.-C. Cai, "The use of pointwise interpolation in domain decomposition with non-nested meshes," *SIAM J. Sci. Comput.*, **14** pp. 250-256 (1995).

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

[4] 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, p. 3. (1990).

[5] 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** p. 401-406 (1994).

[6] P.F. Fischer, "Parallel domain decomposition for incompressible fluid dynamics" *Contemporary Mathematics* **157** AMS p.313-322 (1994).

[7] 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., (1996) p. 595-604.

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

[9] 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.

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

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

[12] J.L. Gustafson, G.R. Montry, and R.E. Benner, *SIAM J. on Sci. and Stat. Comput. 9*, 4 609 (1988).

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

[14] R. van de Geijn, "On global Combine Operations," *Journal of Parallel and Distributed Computing*, **22**, pp. 324-328 (1994).