Parallel Matrix Distributions: Have we been doing it all wrong?

Carter Edwards
Po Geng
Abani Patra

Texas Institute for Computational and Applied Mathematics
The University of Texas at Austin
Austin, TX 78712

and

Robert van de Geijn

Department of Computer Sciences
and
Texas Institute for Computational and Applied Mathematics
The University of Texas at Austin
Austin, TX 78712

Oct. 10, 1995

Abstract

The basic premise of this report is that traditional matrix distributions for distributing matrices on distributed memory parallel architectures are in practice too restrictive. The primary problem lies with the fact that such distributions start with the matrix, not with the underlying physical problem. Through a series of examples, we show how this hampers convenient interfaces between applications and libraries. In some instances, we show how it hampers performance in general. We propose a new data distribution, *Physically Based Matrix Distributions*, which appear to show promise for solving the encountered problems. Some traditionally used distributions are shown to be a special, but often unnatural, case of this more general class of distributions.

1 Introduction

Ever since the conception of distributed memory parallel computing, the problem of distributing data to the individual processors of a parallel computer has been of concern. Perhaps the longest studied problem has been that of distributing matrices to processors. For most dense linear algebra problems, so-called two dimensional data distributions have been shown to be required to obtain scalable high performance [4, 9, 15]. However, some disturbing observations made in this paper seem to indicate the final solution has not yet been found.

The most immediate observation is that the distribution of matrices is typically decoupled from the partitioning and distribution of the underlying physical problem. Indeed, it may stand in the way of convenient interfaces between applications and libraries. Moreover, the distributions for dense matrices fall totally short when applied to common sparse problems, whether they are solved iteratively or directly. We argue how a *physically based matrix distribution* (PBMD) shows promise for solving many of these problems.

The primary intent of this paper is to raise a flag that the final chapter on data distributions has not yet been written. It is not that for *some* applications we have encountered mismatches between application and parallel library interface. It is that for *all* real applications we have pursued, we have encountered some mismatch.

We are not the first to indicate that making the linear operator represented by a matrix the center of the universe leads to a very limited view of the world. Indeed, Alan Edelman quite bluntly states [11]

> **The "All large dense matrices are structured" hypothesis:** This point of view states that nature is not so perverse as to throw n^2 numbers at us haphazardly. Therefore when faced with large n by n matrices, we ought to try our hardest to take advantage of the structure that is surely there.

Matrices encountered in practical computations often result from discretization of differential equation operators. These operators possess the property of local action, i.e., interactions described in matrices are only among physically connected particles/subdomains. Alternative, long range interactions can be approximated using multipole expansions. Both of these methods benefit from focusing on the physical problem first and taking advantage of physical information in structuring the matrix. We believe that this paper is the first paper that, through examples, links the philosophy that matrices are not at the center of the universe to parallel matrix distributions, leading to our Fundamental Principle of Physically Based Matrix Distributions in the next section.

We start our paper by giving traditional approaches to distributing matrices, as well as introducing the class of physically based matrix distributions. Next, we give a number of case studies that illustrate of how traditional data distributions fall short. Furthermore, we illustrate how PBMD may resolve some aspects of the encountered problems. In Section 2.1, we show how traditional methods are in some sense special cases of PBMD. In the conclusion, we hint at other applications where PBMD is more natural.

2 Parallel Matrix Distributions

Parallel matrix algorithms generally depend on the ability to view the p processors in the network as a logical $r \times c$ (two dimensional) array, with $p = rc$. We will denote the (i,j)th processor in this two dimensional mesh as $\mathbf{P}_{i,j}$. Notice that by setting $r = 1$ or $c = 1$, we automatically capture one dimensional meshes (linear arrays) in this model.

2.1 Traditional distributions

In traditionally used distributions, the matrix is partitioned and assigned to processors in one of the three ways: Blocked (Fig. 1), wrapped or cyclic (Fig. 2), or block-wrapped (Fig. 3).

2.2 Physically based matrix distributions

We postulate that one should never start by considering how to decompose the matrix. Rather, one should start by considering how to decompose the physical problem to be solved. Notice that it is the *elements of vectors* that are typically associated with data of physical significance and it is therefore their distribution to processors that is related to the distribution of the problem to be solved. A matrix merely represents the relation between two vectors:

$$y = Ax \tag{1}$$

Since it is more natural to start with distributing the problem to processors, we partition x and y and assign portions of these vectors to processors. We will call a matrix distribution physically based

$$A = \left(\begin{array}{c|c|c|c} A_{0,0} & A_{0,1} & \cdots & A_{0,c-1} \\ \hline A_{1,0} & A_{1,1} & \cdots & A_{1,c-1} \\ \hline \vdots & \vdots & & \vdots \\ \hline A_{r-1,0} & A_{r-1,1} & \cdots & A_{r-1,c-1} \end{array}\right)$$

Figure 1: Blocked distribution: $A_{i,j}$ assigned to $\mathbf{P}_{i,j}$.

$$A = \left(\begin{array}{c|c|c|c} a_{00} & a_{01} & \cdots & a_{0(n-1)} \\ \hline a_{10} & a_{11} & \cdots & a_{1(n-1)} \\ \hline \vdots & \vdots & & \vdots \\ \hline a_{(n-1)0} & a_{(n-1)1} & \cdots & a_{(n-1)(n-1)} \end{array}\right)$$

Figure 2: Wrapped (cyclic) distribution: a_{ij} assigned to $\mathbf{P}_{(i \bmod r, j \bmod c)}$.

$$A = \left(\begin{array}{c|c|c|c} A_{0,0} & A_{0,1} & \cdots & A_{0,N-1} \\ \hline A_{1,0} & A_{1,1} & \cdots & A_{1,N-1} \\ \hline \vdots & \vdots & & \vdots \\ \hline A_{N-1,0} & A_{N-1,1} & \cdots & A_{N-1,N-1} \end{array}\right)$$

Figure 3: Block-wrapped (block-cyclic) distribution: $A_{i,j}$ assigned to $\mathbf{P}_{i \bmod r, j \bmod c}$.

if the layout of the elements of the vectors, x and y, is dictated by the layout of the corresponding physical components.

Notice from Eqn. (1) that rows and columns of matrix A are associated with corresponding elements of y and x, respectively.

> **Fundamental Principle of Physically Based Matrix Distributions:** It is assumed that elements of x and y are distributed to processors according to the natural physical layout of components of the physical problem to processors. If it is convenient for the physical application to use the representation of a two-dimensional mesh and matrix, then columns of A should be assigned to the same column of processors as corresponding elements of x and rows of A should be assigned to the same column of processors as corresponding elements of y.

A particularly important instance of this is the case where x and y are distributed identically. Applications include N-body problems (computation of the force on the particles due to other particles) [14] and iterative methods (conjugate gradient-like iterations require inner-products of x and y) [1].

2.2.1 Simple case

To start our explanation of what distributions meet the conditions of the above principle, we will assume x and y are partitioned into p approximately equal subvectors:

$$x = \left(\begin{array}{c} x_0 \\ \hline x_1 \\ \hline \vdots \\ \hline x_{p-1} \end{array} \right) \text{ and } y = \left(\begin{array}{c} y_0 \\ \hline y_1 \\ \hline \vdots \\ \hline y_{p-1} \end{array} \right)$$

Distribute these subvectors both column-major:

$$\begin{array}{|c|c|c|c|} \hline x_0 & x_r & \cdots & x_{(c-1)r} \\ \hline x_1 & x_{r+1} & \cdots & x_{(c-1)r+1} \\ \hline \vdots & \vdots & & \vdots \\ \hline x_r - 1 & x_{2r-1} & \cdots & x_{p-1} \\ \hline \end{array} \text{ and } \begin{array}{|c|c|c|c|} \hline y_0 & y_r & \cdots & y_{(c-1)r} \\ \hline y_1 & y_{r+1} & \cdots & y_{(c-1)r+1} \\ \hline \vdots & \vdots & & \vdots \\ \hline y_r - 1 & y_{2r-1} & \cdots & y_{p-1} \\ \hline \end{array}$$

where the (i, j) box in the mesh indicates processor $\mathbf{P}_{i,j}$ and the contents of the box indicate the data assigned to that processor.

Letting

$$A = \left(\begin{array}{c|c|c|c} A_{0,0} & A_{0,1} & \cdots & A_{0,p-1} \\ \hline A_{1,0} & A_{1,1} & \cdots & A_{1,p-1} \\ \hline \vdots & \vdots & & \vdots \\ \hline A_{p-1,0} & A_{p-1,1} & & A_{p-1,p-1} \end{array} \right)$$

Our Fundamental Principle, together with the distribution of x and y *induces* the matrix distribution given in Fig. 4. The distribution assigns blocks of columns of the matrix to columns of processors, and *wraps* blocks of rows of the matrix to rows of processors.

2.2.2 Example

We now give an example of the simple case described above, on a six processor mesh.

We start with a 3×2 column major indexing of six processors

$P_{0 \leftrightarrow (0,0)}$	$P_{3 \leftrightarrow (0,1)}$
$P_{1 \leftrightarrow (1,0)}$	$P_{4 \leftrightarrow (1,1)}$
$P_{2 \leftrightarrow (2,0)}$	$P_{5 \leftrightarrow (2,1)}$

We begin by partitioning and distributing the vectors y and x (where $y = Ax$) among the processors. In this example y and x are distributed identically.

$x_0, y_0 \leftrightarrow \mathbf{P}_{0,0}$	$x_3, y_3 \leftrightarrow \mathbf{P}_{0,1}$
$x_1, y_1 \leftrightarrow \mathbf{P}_{1,0}$	$x_4, y_4 \leftrightarrow \mathbf{P}_{1,1}$
$x_2, y_2 \leftrightarrow \mathbf{P}_{2,0}$	$x_5, y_5 \leftrightarrow \mathbf{P}_{2,1}$

We now concentrate on the assignment of the corresponding blocks of the matrix to processors. From the distribution of b we induce the following assignment of matrix blocks to rows of processors:

$$\begin{pmatrix} y_{0\mapsto\mathbf{P}_{0,0}} \\ y_{1\mapsto\mathbf{P}_{1,0}} \\ y_{2\mapsto\mathbf{P}_{2,0}} \\ y_{3\mapsto\mathbf{P}_{0,1}} \\ y_{4\mapsto\mathbf{P}_{1,1}} \\ y_{5\mapsto\mathbf{P}_{2,1}} \end{pmatrix} \Rightarrow \begin{pmatrix} A_{0,0\mapsto\mathbf{P}_{0,*}} & A_{0,1\mapsto\mathbf{P}_{0,*}} & A_{0,2\mapsto\mathbf{P}_{0,*}} & A_{0,3\mapsto\mathbf{P}_{0,*}} & A_{0,4\mapsto\mathbf{P}_{0,*}} & A_{0,5\mapsto\mathbf{P}_{0,*}} \\ A_{1,0\mapsto\mathbf{P}_{1,*}} & A_{1,1\mapsto\mathbf{P}_{1,*}} & A_{1,2\mapsto\mathbf{P}_{1,*}} & A_{1,3\mapsto\mathbf{P}_{1,*}} & A_{1,4\mapsto\mathbf{P}_{1,*}} & A_{1,5\mapsto\mathbf{P}_{1,*}} \\ A_{2,0\mapsto\mathbf{P}_{2,*}} & A_{2,1\mapsto\mathbf{P}_{2,*}} & A_{2,2\mapsto\mathbf{P}_{2,*}} & A_{2,3\mapsto\mathbf{P}_{2,*}} & A_{2,4\mapsto\mathbf{P}_{2,*}} & A_{2,5\mapsto\mathbf{P}_{2,*}} \\ A_{3,0\mapsto\mathbf{P}_{0,*}} & A_{3,1\mapsto\mathbf{P}_{0,*}} & A_{3,2\mapsto\mathbf{P}_{0,*}} & A_{3,3\mapsto\mathbf{P}_{0,*}} & A_{3,4\mapsto\mathbf{P}_{0,*}} & A_{3,5\mapsto\mathbf{P}_{0,*}} \\ A_{4,0\mapsto\mathbf{P}_{1,*}} & A_{4,1\mapsto\mathbf{P}_{1,*}} & A_{4,2\mapsto\mathbf{P}_{1,*}} & A_{4,3\mapsto\mathbf{P}_{1,*}} & A_{4,4\mapsto\mathbf{P}_{1,*}} & A_{4,5\mapsto\mathbf{P}_{1,*}} \\ A_{5,0\mapsto\mathbf{P}_{2,*}} & A_{5,1\mapsto\mathbf{P}_{2,*}} & A_{5,2\mapsto\mathbf{P}_{2,*}} & A_{5,3\mapsto\mathbf{P}_{2,*}} & A_{5,4\mapsto\mathbf{P}_{2,*}} & A_{5,5\mapsto\mathbf{P}_{2,*}} \end{pmatrix}$$

Similarly, from the distribution of x we induce the following assignment of matrix blocks to columns of processors:

$$\begin{pmatrix} A_{0,0\mapsto\mathbf{P}_{*,0}} & A_{0,1\mapsto\mathbf{P}_{*,0}} & A_{0,2\mapsto\mathbf{P}_{*,0}} & A_{0,3\mapsto\mathbf{P}_{*,1}} & A_{0,4\mapsto\mathbf{P}_{*,1}} & A_{0,5\mapsto\mathbf{P}_{*,1}} \\ A_{1,0\mapsto\mathbf{P}_{*,0}} & A_{1,1\mapsto\mathbf{P}_{*,0}} & A_{1,2\mapsto\mathbf{P}_{*,0}} & A_{1,3\mapsto\mathbf{P}_{*,1}} & A_{1,4\mapsto\mathbf{P}_{*,1}} & A_{1,5\mapsto\mathbf{P}_{*,1}} \\ A_{2,0\mapsto\mathbf{P}_{*,0}} & A_{2,1\mapsto\mathbf{P}_{*,0}} & A_{2,2\mapsto\mathbf{P}_{*,0}} & A_{2,3\mapsto\mathbf{P}_{*,1}} & A_{2,4\mapsto\mathbf{P}_{*,1}} & A_{2,5\mapsto\mathbf{P}_{*,1}} \\ A_{3,0\mapsto\mathbf{P}_{*,0}} & A_{3,1\mapsto\mathbf{P}_{*,0}} & A_{3,2\mapsto\mathbf{P}_{*,0}} & A_{3,3\mapsto\mathbf{P}_{*,1}} & A_{3,4\mapsto\mathbf{P}_{*,1}} & A_{3,5\mapsto\mathbf{P}_{*,1}} \\ A_{4,0\mapsto\mathbf{P}_{*,0}} & A_{4,1\mapsto\mathbf{P}_{*,0}} & A_{4,2\mapsto\mathbf{P}_{*,0}} & A_{4,3\mapsto\mathbf{P}_{*,1}} & A_{4,4\mapsto\mathbf{P}_{*,1}} & A_{4,5\mapsto\mathbf{P}_{*,1}} \\ A_{5,0\mapsto\mathbf{P}_{*,0}} & A_{5,1\mapsto\mathbf{P}_{*,0}} & A_{5,2\mapsto\mathbf{P}_{*,0}} & A_{5,3\mapsto\mathbf{P}_{*,1}} & A_{5,4\mapsto\mathbf{P}_{*,1}} & A_{5,5\mapsto\mathbf{P}_{*,1}} \end{pmatrix} \Leftarrow \begin{pmatrix} x_{0\mapsto\mathbf{P}_{0,0}} \\ x_{1\mapsto\mathbf{P}_{1,0}} \\ x_{2\mapsto\mathbf{P}_{2,0}} \\ x_{3\mapsto\mathbf{P}_{0,1}} \\ x_{4\mapsto\mathbf{P}_{1,1}} \\ x_{5\mapsto\mathbf{P}_{2,1}} \end{pmatrix}$$

Collecting all the submatrices to the nodes that own them, we obtain the following matrix distribution:

$\begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} \\ A_{3,0} & A_{3,1} & A_{3,2} \end{pmatrix} \mapsto \mathbf{P}_{0,0}$	$\begin{pmatrix} A_{0,3} & A_{0,4} & A_{0,5} \\ A_{3,3} & A_{3,4} & A_{3,5} \end{pmatrix} \mapsto \mathbf{P}_{0,1}$
$\begin{pmatrix} A_{1,0} & A_{1,1} & A_{1,2} \\ A_{4,0} & A_{4,1} & A_{4,2} \end{pmatrix} \mapsto \mathbf{P}_{1,0}$	$\begin{pmatrix} A_{1,3} & A_{1,4} & A_{1,5} \\ A_{4,3} & A_{4,4} & A_{4,5} \end{pmatrix} \mapsto \mathbf{P}_{1,1}$
$\begin{pmatrix} A_{2,0} & A_{2,1} & A_{2,2} \\ A_{5,0} & A_{5,1} & A_{5,2} \end{pmatrix} \mapsto \mathbf{P}_{2,0}$	$\begin{pmatrix} A_{2,3} & A_{2,4} & A_{2,5} \\ A_{5,3} & A_{5,4} & A_{5,5} \end{pmatrix} \mapsto \mathbf{P}_{2,1}$

2.2.3 General case

Naturally, we may wish to distribute both x and y identically, but in a more complex fashion. For example, let the vectors be partitioned into smaller subvectors for load balancing and wrapped in column-major order:

$x_0, x_p, \ldots$	$x_r, x_{p+r}, \ldots$	$\cdots$	$x_{(c-1)r}, x_{p+(c-1)r}, \ldots$
$x_1, x_{p+1}, \ldots$	$x_{r+1}, x_{p+r+1}, \ldots$	$\cdots$	$x_{(c-1)r+1}, x_{p+(c-1)+1}, \ldots$
$\vdots$	$\vdots$		$\vdots$
$x_{r-1}, x_{p+r-1}, \ldots$	$x_{2r-1}, x_{p+2r-1}, \ldots$	$\cdots$	$x_{p-1}, x_{2p-1}, \ldots$

This induces a matrix distribution that wraps blocks of both columns and rows of the matrix to columns and rows of processors, respectively. The wrapping of rows is *tighter* (smaller blocks) than the wrapping of the columns. Notice that such a distribution of x and y may result from an effort to load balance work associated with the generation of the matrix.

In general, if P is a permutation so that

$$Px = \begin{pmatrix} \tilde{x}_0 \\ \hline \tilde{x}_1 \\ \hline \vdots \\ \hline \tilde{x}_{p-1} \end{pmatrix} \text{ and } Py = \begin{pmatrix} \tilde{y}_0 \\ \hline \tilde{y}_1 \\ \hline \vdots \\ \hline \tilde{y}_{p-1} \end{pmatrix}$$

$$
\begin{array}{|ccc|c|ccc|}
\hline
A_{0,0} & \cdots & A_{0,r-1} & \cdots & A_{0,(c-1)r} & \cdots & A_{0,p-1} \\
A_{r,0} & \cdots & A_{r,r-1} & \cdots & A_{r,(c-1)r} & \cdots & A_{r,p-1} \\
\vdots & & \vdots & & \vdots & & \vdots \\
A_{(c-1)r,0} & \cdots & A_{(c-1)r,r-1} & \cdots & A_{(c-1)r,(c-1)r} & \cdots & A_{(c-1)r,p-1} \\
\hline
A_{1,0} & \cdots & A_{1,r-1} & \cdots & A_{1,(c-1)r} & \cdots & A_{1,p-1} \\
A_{r+1,0} & \cdots & A_{r+1,r-1} & \cdots & A_{r+1,(c-1)r} & \cdots & A_{r+1,p-1} \\
\vdots & & \vdots & & \vdots & & \vdots \\
A_{(c-1)r+1,0} & \cdots & A_{(c-1)r+1,r-1} & \cdots & A_{(c-1)r+1,(c-1)r} & \cdots & A_{(c-1)r+1,p-1} \\
\hline
\vdots & & \vdots & & \vdots & & \vdots \\
\hline
A_{r-1,0} & \cdots & A_{r-1,r-1} & \cdots & A_{r-1,(c-1)r} & \cdots & A_{r-1,p-1} \\
A_{2r-1,0} & \cdots & A_{2r-1,r-1} & \cdots & A_{2r-1,(c-1)r} & \cdots & A_{2r-1,p-1} \\
\vdots & & \vdots & & \vdots & & \vdots \\
A_{p-1,0} & \cdots & A_{p-1,r-1} & \cdots & A_{p-1,(c-1)r} & \cdots & A_{p-1,p-1} \\
\hline
\end{array}
$$

Processor $\mathbf{P}_{i,j}$ receives submatrix

$$
B_{i,j} = \begin{pmatrix}
A_{i,jr} & A_{i,jr+1} & \cdots & A_{i,(j+1)r-1} \\
A_{r+i,jr} & A_{r+i,jr+1} & \cdots & A_{r+i,(j+1)r-1} \\
\vdots & \vdots & & \vdots \\
A_{(c-1)r+i,jr} & A_{(c-1)r+i,jr+1} & \cdots & A_{(c-1)r+i,(j+1)r-1}
\end{pmatrix}
$$

Figure 4: Matrix distribution induced when x and y are subdivided into p subvectors and distributed identically.

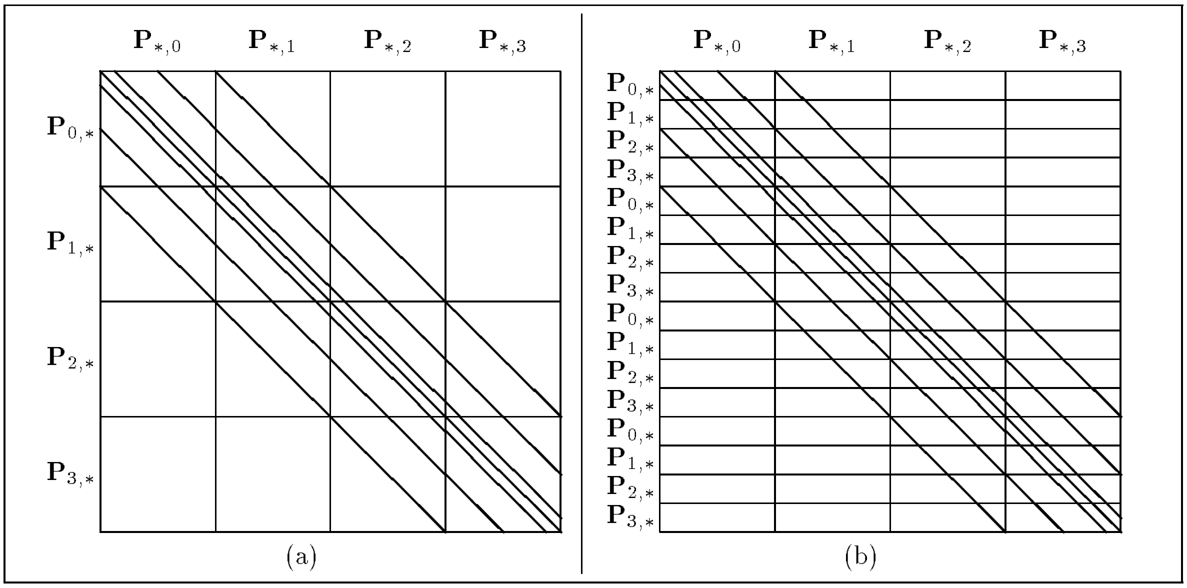

Figure 5: Blocked (left) vs. PBMD (right) distributions. The pictures above represent matrices that have been decomposed as described for these methods. The thick lines indicate structured nonzeroes. For such structured problems, the former leads to severe work imbalance, leaving some processors potentially idle, while the latter distributed the nonzeroes, and the associated work, more effectively.

and $\tilde{x}_i$ and $\tilde{y}_i$ are assigned in column major order, the induced matrix A is assigned to processors as in Fig. 4, except that

$$PAP^T = \left(\begin{array}{c|c|c|c} A_{0,0} & A_{0,1} & \cdots & A_{0,p-1} \\ \hline A_{1,0} & A_{1,1} & \cdots & A_{1,p-1} \\ \hline \vdots & \vdots & & \vdots \\ \hline A_{p-1,0} & A_{p-1,1} & & A_{p-1,p-1} \end{array} \right)$$

An important consequence of using PBMD with x and y distributed identically is

Observation: Given integer constant C, $0 \leq C < n$ and that all subvectors of x and y are of equal length, the set of elements of A, $\{ \ a_{i,j}$ s.t. $((i - j + n) \bmod n) = C \ \}$ is distributed evenly among all processors. In particular, the diagonal is evenly distributed among all processors.

This is illustrated in Fig. 5.

3 A Survey of Experience with Applications

In this section, we outline a number of problem we have encountered when applying traditional data distributions to various algorithms and/or applications. For each, we indicate how PBMD provide a solution to these problems.

3.1 Matrix-vector multiplication

Matrix-vector multiplication is one of the easiest to describe matrix operations: Given vector x of length n and $m \times n$ matrix A, y is formed by $y = Ax$. For simplicity, we will assume $m = n$.

3.1.1 Application: NAS Parallel CG Benchmark

The NAS parallel CG benchmark [1, 2] can be roughly described as a problem that uses an inverse power iteration to find the smallest eigenvalue of a randomly sparse symmetric positive definite matrix. To solve the associated linear system, a simple, unpreconditioned, conjugate gradient method is used. The bulk of the computation is in the sparse matrix-vector multiply and inner-products of the conjugate gradient iteration. The constraint is thus that both x and y must be identically distributed across all processors, to facilitate the inner-products.

3.1.2 Parallel Matrix-Vector Multiplication

A typical implementation [1, 14] will use a blocked matrix distribution as in Fig. 1. The issues for the other traditional distributions are essentially the same. Assuming x is distributed in column-major order, the columns of A assigned to column j of processors is determined by the elements of subvectors $x_{(j-1)r}, \ldots x_{jr-1}$.

3.1.3 Problems

The distribution leads to two problems [14, 16, 17]:

- After the matrix-vector multiplication, the elements of y will invariably be distributed different than x. After all, the elements of subvectors $y_{(i-1)c}, \ldots, y_{ic-1}$, which must eventually reside in processor column i, are computed in processor *row* i, requiring an extra communication to redistribute the subvectors.
- Although the matrix is randomly sparse, it must have a dense diagonal, which is not evenly distributed among all processors.
- In *practical* problems, the sparsity may be "random" in the sense that no advantage can be taken of the sparsity structure. However, nonzero elements can be expected to be concentrated around the diagonal and off-diagonal bands. These regions of the matrix are not distributed evenly among processors.

3.1.4 Benefits of PBMD

In [16, 17], we show how a PBMD induced by distributing x and y identically overcomes all of the concerns mentioned above:

- No additional communication is required since the parallel implementation can be arranged to leave the result y distributed like x.
- The diagonal is distributed equally to all processors.
- The distribution does a better job of distributing bands.
- If necessary, a tighter wrapping of the vectors can be used to improve the distribution of off-diagonal bands.

3.2 Dense linear solver

We wish to compute the solution x to the system of equations $Ax = y$, where A is a dense $n \times n$ matrix. This is typically accomplished by first computing the LU factorization of A:

$$PA = LU$$

where L and U are lower and upper triangular, respectively. P is a permutation matrix that represents the accumulation of all row pivots required for stability (we assume the factorization uses partial pivoting.)

3.2.1 Application: boundary element problems in acoustics

One of the primary applications of parallel dense linear solvers for very large problems come from boundary integral formulations in electromagnetics and acoustics [6, 13]. We will discuss the latter application.

3.2.2 Parallel linear solver implementation

The standard matrix distributions used by parallel dense linear solver packages are two-dimensional (block) wrapped distributions [4, 9, 15, 18]. The benefit of blocking is that it allows the parallel implementation to be more conveniently implemented using level-3 BLAS [8] matrix-matrix operations) which reduce memory traffic on each processor, thereby yielding higher performance. The wrapping is necessary to maintain reasonable load balance as the LU factorization proceeds: During the later stages, computation involves only trailing submatrices of the original matrix, which are not properly balanced among processors if blocked distributions are used.

3.2.3 Problems

There are two fundamental problems with dense linear solver packages based on traditional data distributions:

- While there is no sparsity, the work associated in generating the matrix is often not equal for all parts of the matrix.

  A special difficulty in the acoustics application is that the integral equation is not uniquely solvable at certain frequencies. This non-uniqueness problem is overcome by combining the original integral formulation with a hypersingular integral formulation (the Burton-Miller method). The whole integral formulation then is approximated by the Galerkin method. The hypersingular integrals are avoided through a special transformation on the weak formulation [13]. The Burton-Miller formulation induces significant extra work in generating the dense matrices characteristic of integral formulations. Generating blocks of the matrix on the diagonal proved substantially more expensive than blocks away from the diagonal. This was due to the fact that more elements of the matrix in blocks on the diagonal of the matrix must be computed through singular integrals. In order to perform singular integrals (either by the Duffy triangular coordinate method [7] or the local polar coordinate method [3]), the original element must be divided into several subtriangles; and then the integrals are performed separately in all subtriangles. This procedure is much more expensive than the regular Gauss quadrature integral performed in one element, and so, unless the logical mesh of P processes is in the form of $1 \times P$ or $P \times 1$, some load imbalance will be created.

  Naturally, one can argue that filling the matrix is an $O(n^2)$ operation, and hence lower order compared to the $O(n^3)$ operations required to perform the solve. Nonetheless, for many applications, parallel generation of the matrix requires time comparable to the solve. *Indeed, in our study, it became beneficial to generate the matrix using a one-dimensional data distribution ($r = 1$), writing the matrix to disk, and reloading it using the two-dimensional data distribution required by the dense solver.*

- The second problem is more frustrating for those of us who develop libraries than for the user of the library: Having had extensive experience designing and implementing parallel dense linear packages, using techniques also used by packages like ScaLAPACK, we have faced a fundamental problem: How do we distribute the vector? The question has always been whether the vector should be distributed like rows of a matrix, or like columns, or like diagonals? In any of these solutions, only a small subset of processors hold the vector.

3.2.4 Benefits of PBMD

Physically based matrix distributions retain the benefits of wrapped data distributions. By decomposing the vectors into more subvectors than processors, and either row or column wrapping, the matrix is wrapped, although tighter in one dimension than the other.

Although we have yet to implement a dense solver package using this data distribution, we have many years of experience with more traditional data distributions. It is thus our belief that little performance degradation will result. The benefits due to simpler library interfaces will likely outweigh any performance degradation.

3.3 Factorization of block-sparse matrices

We wish to compute the solution x to the system of equations $Ax = y$, where A is a sparse $n \times n$ matrix, this can be accomplished by first factoring

$$A = LU$$

where L and U are lower and upper triangular, respectively. By taking advantage of sparsity, computational and storage requirements can be greatly reduced. The sparsity is usually a result of the local action of the underlying operator.

3.3.1 Application: hp-adaptive FEM problems

A standard approach to solving partial differential equations arising in engineering and physics is to discretize the problem using finite element methods. The most sophisticated of these use highly adaptive hp meshes, wherein both the local element size and the polynomial order are dynamically chosen for maximum efficiency [19]. These problems lead to highly irregular sparse linear systems. However, the sparsity typically exhibits itself as locally dense blocks. The sparsity pattern is dictated by both the connectivity of the graph associated with the discretization mesh and the local polynomial distributions. A linear element (polynomial degree one) on a three dimensional scalar problem has eight nonzeroes per row, whereas a fifth order element on a three dimensional scalar problem would lead to 216 nonzeroes per row. Realistic problems consist of three to four degrees of freedom per node, multiplying the number of nonzeroes accordingly.

3.3.2 Parallel implementation

Methods proposed for parallel implementations of direct LU factorization and corresponding sparse triangular solvers take advantage of local density with techniques like nested disection and recursive spectral bisection orderings. These methods create a physically based ordering that reduces the amount of fill-in that occurs during the factorization stage, thereby reducing required computation.

Given an ordering, one effective general purpose implementations of sparse factorization is given by Rothberg and Schreiber [20, 21]. In their implementation, they use a "supernodal" method that allows them to take advantage of dense blocks in the sparse matrix. To distribute the sparse matrix among processors, they view the processors as a logical two dimensional mesh and a block-wrapped data distribution, *except that they use a heuristic that maps the columns and rows asymmetrically.* Intuitively, their approach makes sense, since eventually fill-in leads to requirements during the later stages of the algorithm that are much like those of dense matrices. However, during the earlier stages, the diagonal needs to be distributed among all nodes.

3.3.3 Problems

By design, a primary source of density in the matrix is along the diagonal of the matrix. Indeed, the closer to diagonal the matrix, the less fill-in occurs. As with the dense linear solve and the sparse matrix-vector multiply, this leads to an imbalance in generating the problem and/or initial stages of the factorization itself. This is particularly problematic, since clever orderings allow a lot of parallelism early in the computation. Indeed, during these early stages, ideally we want the portion of the matrix being factored to be *block diagonal*, with individual blocks assigned to different processors.

Rothberg and Schreiber overcome this problem by using a heuristic that maps rows and columns in a nonsymmetric fashion. This then spreads out the diagonal among the processors.

3.3.4 Benefits of PBMD

The implementation of hp-adaptive FEM solvers is a prime example of how PBMD goes naturally with the distribution of the physical problem: Elements are distributed to processors in an effort to achieve load balance during the generation and solving of the linear system. In addition, careful ordering of the elements reduces both matrix fill-in and communication during the parallel factorization.

Hence, we assign subvectors (corresponding to physical partitions) in some prescribed fashion to processors. However, to further reduce fill-in in the matrix, each subvector can be further partitioned into two – a set of exterior variables and a set of interior variables $y_k = \{y_{kE}, y_{kI}\}$. The interior variables y_{kI} have no non-zero interactions with any variable outside the partition. The immediate implication is that the dense diagonal block associated with this can be eliminated locally with no resulting fill in outside the block. Furthermore, since the exterior of each physical partition is of a space dimension one less that the original space dimension (surface instead of volume), the number of variables in the exterior partition is significantly smaller than the interior partition. Hence most of the work is done on blocks of the matrix along the diagonal before any fill-in or communication needs occur. Traditional two-dimensional distributions would assign this work to only a few processors. Nonetheless, once the fill-in does generate a nearly dense trailing submatrix, a two dimensional data distribution can be ensured using some form of wrapping. In essence, PBMD reaches the goal of the heuristic used by Rothberg and Schreiber in a more organized fashion, based on solid principles.

The conclusion is that from the physical attributes of the problem, very intelligent decisions can be made about how to distribute the physical problems. While traditional data distributions can get in the way of library routines that can then be used to solve the associated linear systems, PBMD appears to fit the requirements naturally.

4 Traditional methods: a Special Case

In this section, we show how traditional matrix distributions can be viewed as a special case of physically based matrix distributions, when x and y are distributed appropriately. By our fundamental principle, *if* x and y are naturally distributed as required, the traditional methods are a special case of PBMD. Otherwise, they are an unnatural case of PBMD.

4.1 Inducing a blocked distribution

Let

$$x = \left(\begin{array}{c} x_0 \\ \hline x_1 \\ \hline \vdots \\ \hline x_{p-1} \end{array} \right) \text{ and } y = \left(\begin{array}{c} y_0 \\ \hline y_1 \\ \hline \vdots \\ \hline y_{p-1} \end{array} \right)$$

If we now view the processors logically as a two dimensional array, we must decide how to assign the subvectors of x and y to these processors.

Assign subvectors of x to processors in column-major order and subvectors of y to processors in row-major order:

x_0	x_r	$\cdots$	$x_{(c-1)\times r}$
x_1	x_{r+1}	$\cdots$	$x_{(c-1)\times r+1}$
x_2	x_{r+2}	$\cdots$	$x_{(c-1)\times r+2}$
$\vdots$	$\vdots$		$\vdots$
x_{r-1}	$x_{(2r-1)}$	$\cdots$	x_{p-1}

y_0	y_1	$\cdots$	y_{c-1}
y_c	y_{c+1}	$\cdots$	y_{2c-1}
y_{2c}	y_{2c+1}	$\cdots$	y_{3c-1}
$\vdots$	$\vdots$		$\vdots$
$y_{(r-1)c}$	$y_{(r-1)c+1}$	$\cdots$	y_{p-1}

Consider again the equation $y = Ax$. Notice that elements of y are formed from corresponding rows of A and columns of A are multiplied by corresponding elements of x before being added together to form y. If the matrix is blocked as in Fig. 1 then

$$\begin{pmatrix} y_{ic} \\ \hline y_{ic+1} \\ \hline \vdots \\ \hline y_{(i+1)c-1} \end{pmatrix} = A_{i,0} \begin{pmatrix} x_0 \\ \hline x_1 \\ \hline \vdots \\ \hline x_{r-1} \end{pmatrix} + A_{i,1} \begin{pmatrix} x_r \\ \hline x_{r+1} \\ \hline \vdots \\ \hline x_{2r-1} \end{pmatrix} + \cdots + A_{i,c-1} \begin{pmatrix} x_{(c-1)r} \\ \hline x_{(c-1)r+1} \\ \hline \vdots \\ \hline x_{p-1} \end{pmatrix}$$

Notice that if the dimensions of blocks $A_{i,j}$ are chosen appropriately, elements of vector x exist within the same *column* of processors as $A_{i,j}$, and elements of vector y exist in the same *row* of processors. Turning this observation around, we can say that *if* the elements of x and y are distributed as indicated, then a logical distribution for A is to block it, and assign $A_{i,j}$ to processor $\mathbf{P}_{i,j}$. In other words:

> Partitioning x and y into p subvectors and distributing them in row and column major order, respectively, *induces* a blocked matrix distribution.

4.2 Inducing a wrapped distribution

Let

$$x = \begin{pmatrix} x_0 \\ \hline x_1 \\ \hline \vdots \\ \hline x_{n-1} \end{pmatrix} \text{ and } y = \begin{pmatrix} y_0 \\ \hline y_1 \\ \hline \vdots \\ \hline y_{n-1} \end{pmatrix}$$

View the processors logically as a two dimensional array and assign subvectors of x to processors in row-major order and subvectors of y to processors in column-major order, wrapping if necessary:

$x_0, x_p, \ldots$	$\cdots$	$x_{c-1}, x_{p+c-1}, \ldots$
$x_c, x_{p+c}, \ldots$	$\cdots$	$x_{2c-1}, x_{p+2c-1}, \ldots$
$x_{2c}, x_{p+2c}, \ldots$	$\cdots$	$x_{3c-1}, x_{p+3c-1}, \ldots$
$\vdots$		$\vdots$
$x_{(r-1)c}, x_{p+(r-1)c}, \ldots$	$\cdots$	$x_{p-1}, x_{2p-1}, \ldots$

$y_0, y_p, \ldots$	$\cdots$	$y_{(c-1)\times r}, y_{p+(c-1)\times r}, \ldots$
$y_1, y_{p+1}, \ldots$	$\cdots$	$y_{(c-1)\times r+1}, y_{p+(c-1)\times r+1}, \ldots$
$y_2, y_{p+2}, \ldots$	$\cdots$	$y_{(c-1)\times r+2}, y_{p+(c-1)r+2}, \ldots$
$\vdots$		$\vdots$
$y_{r-1}, y_{p+r-1}, \ldots$	$\cdots$	$y_{p-1}, y_{2p-1}, \ldots$

If the matrix is blocked as in Fig. 2, then

$$
\left(\begin{array}{c} y_i \\ \hline y_{r+i} \\ \hline y_{2r+i} \\ \hline \vdots \end{array}\right)
=
\left(\begin{array}{c|c|c|c} a_{i,0} & a_{i,c} & a_{i,2c} & \cdots \\ \hline a_{(r+i),.0} & a_{(r+i),c} & a_{(r+i),2c} & \cdots \\ \hline a_{(2r+i),.0} & a_{(2r+i),c} & a_{(2r+i),2c} & \cdots \\ \hline \vdots & \vdots & \vdots & \ddots \end{array}\right)
\left(\begin{array}{c} x_0 \\ \hline x_c \\ \hline x_{2c} \\ \hline \vdots \end{array}\right)
$$
$$
+ \left(\begin{array}{c|c|c|c} a_{i,1} & a_{i,c+1} & a_{i,2c+1} & \cdots \\ \hline a_{(r+i),.1} & a_{(r+i),c+1} & a_{(r+i),2c+1} & \cdots \\ \hline a_{(2r+i),.1} & a_{(2r+i),c+1} & a_{(2r+i),2c+1} & \cdots \\ \hline \vdots & \vdots & \vdots & \ddots \end{array}\right)
\left(\begin{array}{c} x_1 \\ \hline x_{c+1} \\ \hline x_{2c+1} \\ \hline \vdots \end{array}\right)
$$
$$
+\cdots
$$

Notice that again elements of vector x exist within the same *column* of processors as columns of A and elements of vector y exist in the same *row* of processors as rows of A. Turning this observation around, we can say that *if* the elements of x and y are distributed as indicated, then a logical distribution for A is to wrap it, assigning a_{ij} to processor $\mathbf{P}_{(i \bmod r),(j \bmod c)}$. In other words:

> Partitioning x and y into n elements and wrapping them in row and column major order, respectively, *induces* a two dimensional wrapped matrix distribution.

4.3 Inducing a block-wrapped distribution

Finally, a block-wrapped distribution can be induced by partitioning x and y into small vectors, rather than individual elements, and distributing to processors at that granularity.

5 Conclusions

We have demonstrated encountered problems with traditional matrix distributions and proposed an alternative distribution that appears to solve some of the problems. It is our belief that PBMD is much more natural: it allows the user of a parallel linear algebra library to concentrate on decomposing the problem rather than on how to fit the generation of the problem into a matrix distribution that has little to do with the physical problem. In addition, we have argued that the new data distribution is much more natural for sparse algorithms.

While we have used the problem of linear system solvers to illustrate our point, very similar issues arise in the solution of linear eigenvalue problems. Take, for instance, traditional methods for solving the symmetric algebraic eigenvalue problem: The first phase involves a reduction to tridiagonal form, which requires a two-dimensional data distribution for scalability [5]. However, subsequently, the tridiagonal eigenvalue problem must be solved. While traditional matrix distributions leave the tridiagonal on a small number of processors, PBMD leaves it distributed among all processors.

Matrices can come in many forms, and need not be explicitly formed. Matrices can exist as trees containing information on how to perform the computation (e.g. used by fast multipole methods) or may exist implicitly (e.g. implicit element-by-element finite element methods). In that case, PBMD may give an insight into where the computation that could be expressed as a matrix should be performed.

Acknowledgements

The financial support of DARPA under Contract DABT63-92-C-0042 and the Office of Naval Research under Contract N00014-95-1-0401 is gratefully acknowledged.

The ideas leading to the PBMD started with a collaboration on the implementation of the NAS Parallel CG Benchmark with John Lewis and David Payne. We would like to mention a few people who have worked with us over the years on research that also led up to the presented ideas. These include Tom Cwik, Jim Demmel, Jack Dongarra, Shaoze Ouyang, Jean Patterson, David Payne, Rob Schreiber, Yun Shen. and David Walker. We thank Alan Edelman for suggestions made after reviewing an earlier draft.

References

[1] The NAS Parallel Benchmarks. David Bailey, John Barton, Thomas Lasinski and Horst Simon (editors). NASA Technical Memorandum 103863, NASA Ames Research Center, Moffett Field, CA, July 1993.

[2] D. H. Bailey, E. Barszcz, L. Dagum, and H. D. Simon. NAS Parallel Benchmark Results, Proceedings of SHPCC94.

[3] C.A Brebbia, J. C. F. Telles and L. C. Wrobel 1984, *Boundary Element Techniques, Theory and applications in Engineering*, Springer-Verlag.

[4] J. Choi, J. J. Dongarra, R. Pozo, and D. W. Walker, "Scalapack: A Scalable Linear Algebra Library for Distributed Memory Concurrent Computers, *Proceedings of the Fourth Symposium on the Frontiers of Massively Parallel Computation.* IEEE Comput. Soc. Press, 1992, pp. 120-127.

[5] J. Choi, J. Dongarra, and D. Walker, "The Design of a Parallel Dense Linear Algebra Software Library: Reduction to Hessenberg, Tridiagonal, and Bidiagonal Form," UT, CS-95-275, February 1995.

[6] Tom Cwik, Robert van de Geijn, and Jean Patterson, "Application of Massively Parallel Computation to Integral Equ Models of Electromagnetic Scattering," *Journal of the Optical Society of America A,* Vol. 11, No. 4, April 1994, pp. 1538–1545

[7] L. Demkowicz, A. Karafiat and J.T. Oden 1992 *Comp. Meths. Appl. Mech. Engrg.* **101**, 251-282. Solution of elastic scattering problems in linear acoustics using *h-p* boundary element method.

[8] J. J. Dongarra, J. Du Croz, S. Hammarling, and I. Duff, "A Set of Level 3 Basic Linear Algebra Subprograms," *TOMS*, Vol. 16, No. 1, ppages 1–17, 1990.

[9] Jack. J. Dongarra, Robert A. van de Geijn, and David W. Walker, "Scalability Issues Affecting the Design of a Dense Linear Algebra Library," *Journal of Parallel and Distributed Computing,* Vol. 22, No. 3, Sept. 1994, pp. 523–537.

[10] L. Demkowicz, J. T. Oden, W. Rachowicz and O, Hardy "Toward A Universal *hp* Adaptive Finite Element Strategy, Part 1. Constrained Approximation and Data Structure" , *Comput. Methods. Appl. Mech. and Engg.*, 77(1989), pp.79-112

[11] A. Edelman, "Large Dense Numerical Linear Algebra in 1993: The Parallel Computing Influence". Journal of Supercomputing Applications. 7 (1993), pp. 113–128.

[12] G. Fox, et al., *Solving Problems on Concurrent Processors: Volume 1*, Prentice Hall, Englewood Cliffs, NJ, 1988.

[13] P. Geng, J. T. Oden and R. A. van de Geijn 1995, Massively Parallel Computation for Acoustical Scattering Problems using Boundary Element Methods *to appear in Journal of Sound and Vibration.*

[14] B. Hendrickson, R. Leland, and S. Plimpton, A Parallel Algorithm for Matrix-Vector Multiplication, Tech. Rep. SAND 92-2765, Sandia National Laboratories, Albuquerque, NM, March 1993.

[15] B. A. Hendrickson and D. E. Womble, "The Torus-Wrap Mapping for Dense Matrix Calculations on Massively Parallel Computers," SIAM J. Sci. Comput., **check issue number**

[16] J. G. Lewis, D. G. Payne, and R. A. van de Geijn, "Matrix-Vector Multiplication and Conjugate Gradient Algorithms on Distributed Memory Computers," in *Proceedings of the Scalable High Performance Computing Conference 1994*.

[17] J.G. Lewis and R.A. van de Geijn, Distributed Memory Matrix-Vector Multiplication and Conjugate Gradient Algorithms, in the proceedings of *Supercomputing '93*, Portland, OR, November 15-19, 1993.

[18] W. Lichtenstein and S. L. Johnsson, "Block-Cyclic Dense Linear Algebra", Harvard University, Center for Research in Computing Technology, TR-04-92, Jan., 1992.

[19] J. T. Oden, A. Patra, Y. Feng, "Parallel Domain Decomposition Solver For Adaptive *hp* Methods" submitted to *SIAM Journal for Numerical Methods.*

[20] E. Rothberg and R. Schreiber, "Improved load distribution in parallel sparse Cholesky factorization." in *Proceedings of Supercomputing 94*, pp. 783–792.

[21] E. Rothberg and R. Schreiber, "Efficient parallel sparse Cholesky factorization," in *Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing* (R. Schreiber, *et al.* eds.), SIAM, 1994, pp. 407–412.