LAPACK Working Note 116

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

Majed Sidani and Bill Harrod

Applications Division
Cray Research, A Silicon Graphics Company

November 14, 1996

* Electronic mail address: sidani@cray.com, harrod@cray.com

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

Majed Sidani and Bill Harrod
Applications Division
Cray Research, A Silicon Graphics Company

November 14, 1996

1 Introduction

Providing a framework for matrix distributions in parallel linear algebra libraries that is flexible enough for parallel applications developers is an important feature. The added flexibility could however result in added complexity in the libraries development and present the developers with many challenges.

A new package [6], PLAPACK, containing a number of parallel linear algebra solvers, was brought to our attention as a prototype of a more flexible alternative to ScaLAPACK ([3]). The parallel matrix distribution in PLAPACK, its authors submitted, is the new feature that allowed the superior flexibility where applications developers are concerned. The purpose of this short note is to show that the PLAPACK matrix distribution scheme is a variation on the ScaLAPACK scheme.

The answer to the question we asked in the title is: Yes! What prompted the question – and supports the answer – is the fact that the authors of PLAPACK started their investigation by considering the parallel distributions needs of applications, and as a result determined the distribution to be supported in PLAPACK. The fact that their implementation does not provide any matrix distribution capability beyond what is available in ScaLAPACK is remarkable, given that the authors of PLAPACK's position is that this library is not quite satisfactory ([2]).

2 Block-Cyclic Distributions

We review briefly here the parallel matrix distributions known as block-cyclic distributions. The *npes* processors available to the application are viewed as filling a logical two-dimensional grid of r rows and c columns, with $npes = r * c$. Given a $m \times n$ matrix A, partition it as

$$\begin{pmatrix} A_{00} & \cdots & A_{0q} \\ \vdots & & \vdots \\ A_{p0} & \cdots & A_{pq} \end{pmatrix}$$

where each $A_{i,j}$ is a block of b_r rows and b_c columns, with the possible exception of the blocks in the last column which may have fewer columns than b_c, and the blocks in the last row which may have fewer rows than b_r. Then block $A_{i,j}$ is mapped to the processor in the $(i\ mod\ r,\ j\ mod\ c)$ position of the grid, i.e., the processor in row, $i\ mod\ r$, and column, $j\ mod\ c$. For example, assume that a 2×3 grid of processors is chosen. Then figure 1 illustrates one possible mapping of the six processors to the grid. If $m = 6 * b_r$ and $n = 6 * b_c$, then figure 2 illustrates how the blocks of A are mapped to the grid. By varying the dimensions of the blocks of A and those of the grid, one can generate different mappings.

0	2	4
1	3	5

Figure 1: *A two-dimensional grid filled in a column major order with 6 processors.*

$$\left(\begin{array}{c|c|c|c|c|c} 0 & 2 & 4 & 0 & 2 & 4 \\ \hline 1 & 3 & 5 & 1 & 3 & 5 \\ \hline 0 & 2 & 4 & 0 & 2 & 4 \\ \hline 1 & 3 & 5 & 1 & 3 & 5 \\ \hline 0 & 2 & 4 & 0 & 2 & 4 \\ \hline 1 & 3 & 5 & 1 & 3 & 5 \end{array}\right)$$

Figure 2: *Block-cyclic distribution of a matrix. A number in a block is that of the processor owning the particular block of the matrix.*

3 PBMD

The Physically Based Matrix Distribution (PBMD) was proposed by the authors of PLAPACK [6, 1, 2] as a basis for more flexible parallel linear algebra libraries. It was claimed that applications developers will not have to "unnaturally" modify the way they want to distribute data in order to fit the library's required distribution. They have the freedom to distribute the vectors (which contain the data of "physical significance") across processors in a way that only depends on the application. The matrix (the operator) is then distributed in a conforming way that in effect optimizes matrix-vector products involving the vectors and the matrix. The concept in its most general form will not be discussed in any more details in this note. Interested readers are referred to the articles mentioned above.

We describe instead the Physically Based Matrix Distribution that is supported in PLAPACK. Given a two-dimensional grid of r rows and c columns of processors, a vector (1-dimensional), is assumed to be distributed uniformly by blocks of b components across the processors of the entire grid, in a column-major order. Assume that x and y are vectors of length n that are related by $y = A * x$, where A is a $n \times n$ matrix. Then PLAPACK requires that x and y be distributed conformally in this fashion, and that A be partitioned into square blocks of order b, and distributed according to the following rule: A block of A is in the same column of the processor grid as the block of x it multiplies and the same row of the grid as the block of y it updates. Now, starting from the first block, every r contiguous blocks of x will reside in the same column of the grid of processors. Contiguous blocks of y will reside on different rows of the grid in general ($r > 1$). When the rule above is applied, the blocks of A in one row of blocks are dealt r at a time to processors in a row of the grid, and the blocks in the next row of blocks are assigned to the next row of processors in the grid. If r contiguous blocks of A of order b are viewed as one block of b rows and $r * b$ columns, PBMD is then readily found to be a block-cyclic distribution, with the blocks of A being $b \times r * b$. In the notation of the previous section, PLAPACK's PBMD is a block-cyclic distribution with $b_r = b$ and $b_c = r * b$.

An example follows.

Example: Assume that the (user-defined) grid of processors is the same as in figure 1, which we reproduce here for convenience.

0	2	4
1	3	5

We have then $r = 2$ and $c = 3$ using our notation for the number of rows and columns in the grid.

Let x and y be two vectors of length n that are related by $y = A * x$, where A is a $n \times n$ matrix. As we stated earlier, PLAPACK requires that these vectors be partitioned in a uniform fashion and that the same block size be used for both vectors. Assume that x and y are partitioned into 6 blocks. Let b denote the block size. Then these vectors are mapped to the processor grid as follows:

y_0	y_2	y_4
y_1	y_3	y_5

x_0	x_2	x_4
x_1	x_3	x_5

In general, if x_i is the i^{th} block of x $(i = 0, \cdots)$, and $row(x_i)$ and $col(x_i)$ are the row and column numbers of the processor of the grid to which x_i is mapped, then we have:

$$row(x_i) = i \ mod \ r, \qquad col(x_i) = (i/r) \ mod \ c,$$

where i/r is the euclidean quotient.

Given the partitioning of x and y, A is partitioned into 6 blocks in each dimension, in order to conform to the partitioning of x and y. The blocks of A are square of order b (with the usual possible exceptions for the last blocks in a column or a row).

$$\begin{pmatrix} A_{00} & A_{01} & A_{02} & A_{03} & A_{04} & A_{05} \\ A_{10} & A_{11} & A_{12} & A_{13} & A_{14} & A_{15} \\ A_{20} & A_{21} & A_{22} & A_{23} & A_{24} & A_{25} \\ A_{30} & A_{31} & A_{32} & A_{33} & A_{34} & A_{35} \\ A_{40} & A_{41} & A_{42} & A_{43} & A_{44} & A_{45} \\ A_{50} & A_{51} & A_{52} & A_{53} & A_{54} & A_{55} \end{pmatrix}$$

According to the PBMD distribution rule, block A_{ij} should be mapped to the processor of the grid in the $(row(y_i), col(x_j))$ position, i.e., the $(i \ mod \ r, (j/r) \ mod \ c)$ position. For instance:

A_{00} and A_{01} must reside on processor $(0,0)$, i.e., processor 0 in our grid; A_{02} and A_{03} must reside on processor $(0,1)$, i.e., processor 2, and so on. Figure 3 shows where each block of A in our example resides.

$$\left(\begin{array}{c|c|c|c|c|c}
0 & 0 & 2 & 2 & 4 & 4 \\ \hline
1 & 1 & 3 & 3 & 5 & 5 \\ \hline
0 & 0 & 2 & 2 & 4 & 4 \\ \hline
1 & 1 & 3 & 3 & 5 & 5 \\ \hline
0 & 0 & 2 & 2 & 4 & 4 \\ \hline
1 & 1 & 3 & 3 & 5 & 5
\end{array}\right)$$

Figure 3: *Given the grid in figure 1 and a block size, PLAPACK's PBMD partitions the matrix uniformly into square blocks, and deals the blocks to the processors as shown. The number of contiguous blocks in a row that are mapped to one processor is equal to the number of rows in the grid of processors – in the previous example it is 2. When the contiguous blocks that are residing on one processor are viewed as one block (with 2 times the number of columns), PLAPACK's PBMD reduces to a traditional block-cyclic distribution.*

It is clear that given any k, $A_{i,kr}, A_{i,kr+1}, \cdots, A_{i,kr+(r-1)}$ will reside on the same processor $(i \ mod \ r, k \ mod \ c)$. But these r contiguous blocks can be viewed as one aggregate block with $r*b$ columns, indeed the k^{th} such block in the i^{th} row. Let's call these aggregate blocks $\hat{A}_{i,k}$. Then $\hat{A}_{i,k}$ is mapped to processor $(i \ mod \ r, k \ mod \ c)$. This shows that the parallel matrix distribution in PLAPACK is no more than a block-cyclic distribution of these aggregate blocks[1].

4 Algorithmic Issues

It is natural to ask whether this specific block-cyclic distribution has resulted in any algorithmic advantages for parallel linear algebra operations. In this section, we will examine the impact of this distribution on basic algorithms.

[1]High Performance FORTRAN (HPF, [4]) provides a compact notation in this context. In HPF notation, PLAPACK requires that vectors and matrices be distributed as:

```
!HPF$ TEMPLATE, DIMENSION( r * c ) :: GRID1
!HPF$ TEMPLATE, DIMENSION( r, c ) :: GRID2
!HPF$ DISTRIBUTE x( BLOCK(b) ), y( BLOCK(b) ) ONTO GRID1
!HPF$ DISTRIBUTE A( BLOCK(b), BLOCK(r * b) ) ONTO GRID2
```

Using our example in the previous section, assume that we want to compute $y = A * x$. Then the first step in the PLAPACK implementation of this operation is to perform the following data communications:

$$x_0 \rightarrow (1,0) \qquad x_2 \rightarrow (1,1) \qquad x_4 \rightarrow (1,2)$$

$$x_1 \rightarrow (0,0) \qquad x_3 \rightarrow (0,1) \qquad x_5 \rightarrow (0,2)$$

where (i, j) is the processor in the i^{th} row and j^{th} column of the grid. In general, an all-to-all broadcast is performed across each column in the grid which leaves x entirely distributed across each row in the grid.
This is the starting point for the matrix-vector computation in ScaLAPACK (more precisely, ScaLAPACK requires that x be on one row or one column of the grid). And indeed, the matrix-vector product computations in PLAPACK and ScaLAPACK are identical after that point until y is computed on one column of the grid of processors. After that, PLAPACK scatters each block of y across the row of the grid where it resides, which results in y being distributed across all the processors of the grid. This example suggests that no particular advantage was gained from PLAPACK's distribution for this important computation.

More generally, the matrix distribution in PLAPACK is but one instance of a broader class of distributions that is available in ScaLAPACK. The broader class of distributions provides for more flexibility in algorithmic choices and might conceivably allow more efficient algorithms for distributed operations.

Lastly, we consider the impact of this distribution on parallel matrix generation and the claims of the authors of PLAPACK in this regard. Henceforth, the matrix is assumed to be square and we define a "diagonal block" of the matrix to be one containing a portion of the main diagonal of the matrix.
Parallel linear algebra computations consist in general of two stages. First the matrix must be generated. Second, a linear algebra problem, such as solving a linear system or computing an eigensystem is to be addressed. The PLAPACK authors point to an important problem with

regard to the first stage. In some applications, the generation of the diagonal blocks of the matrix may require more intensive computations than the rest of the matrix. Diagonal blocks could require computation involving singular integrals, for example. It is therefore important to ensure that the generation of the diagonal blocks be distributed across as many processors as possible. The PLAPACK authors, however, claim that the diagonal blocks of the matrix are better distributed in a PBMD distribution ([2, 5]) than in other distributions, in the sense that more processors get a piece of the diagonal. In particular, the authors claim that in distributions other than PBMD, the diagonal blocks are distributed across $\sqrt(npes)$ processors only, where $npes$ is the total number of processors. This is not accurate! It is only correct if the matrix is partitioned into square blocks *and* the grid of processors is square.

In general, given a matrix that is partitioned into square blocks and distributed in a block-cyclic fashion across a two-dimensional grid of $r \times c$ processors, the diagonal blocks are distributed across $lcm(r,c)$ processors, the least common multiple of r and c. To see this, note that the diagonal blocks in this case are A_{ii}, $(i = 0, \cdots)$ and that they are mapped to processors $(i \ mod \ r, i \ mod \ c)$ in the grid. The sequence $(i \ mod \ r, i \ mod \ c)$, $(i = 0, \cdots)$, is periodic since

$$(\ (i_1 + lcm(r,c)) \ mod \ r, \ (i_1 + lcm(r,c)) \ mod \ c \ ) = (i_1 \ mod \ r, i_1 \ mod \ c).$$

The number of (distinct) processors that get a diagonal block is equal to the period of this sequence, which we now determine. Let,

$$(i_1 \ mod \ r, i_1 \ mod \ c) = (i_2 \ mod \ r, i_2 \ mod \ c).$$

Then,

$$(i_2 - i_1) \ mod \ r = 0$$

$$(i_2 - i_1) \ mod \ c = 0$$

Thus $i_2 - i_1$ is a multiple of $lcm(r,c)$, which proves our claim. Note that in particular, if r and c are relatively prime then *every* processor in the grid gets some diagonal blocks.

It is also possible to increase the number of processors that get a diagonal block by partitioning the matrix into non-square blocks. To see this, assume for simplicity that, using our notation from section 2, $b_c = l * b_r$ where l is some integer and that the grid of processors is square[2]. It is easy to see that

$$A_{00}, A_{10}, \cdots, A_{l-1,0}, \cdots, A_{l*k,k}, \cdots, A_{l*k+l-1,k}, \cdots$$

are the diagonal blocks of A (i.e., those containing a portion of the main diagonal of A). Since A_{ij} maps to processor $(i\ mod\ r, j\ mod\ c)$ in a block-cyclic distribution,

$$A_{l*k,k}, \cdots, A_{l*k+l-1,k}$$

will map to processors in the same column of the grid, namely $k\ mod\ c$. The number of processors in one column of the grid getting a diagonal block is therefore $min(r, l)$[3]. The total number of processors in the grid getting a diagonal block is therefore

$$min(r, l) * c. \tag{1}$$

Similarly, if $b_r = l * b_c$, then the total number of processors getting a diagonal block is,

$$min(c, l) * r. \tag{2}$$

An appropriate choice of l will distribute the diagonal blocks across more than $\sqrt(npes)$ processors.

We conclude that in order to distribute the diagonal blocks across more than $\sqrt(npes)$ processors, one either chooses a grid of processors with unequal sides or partitions the matrix into non-square blocks. PBMD does the latter. The specific aspect ratio chosen by PBMD is of no relevance; it just has to be different from 1:1.

[2] More precisely, it can be shown that the results (1) and (2) that are derived in this paragraph hold true whenever $l * c\ mod\ r = 0$, or $l * r\ mod\ c = 0$, resp., of which a square grid is a particular case. For this reason we have kept separate notations for r and c.

[3] Our assumptions about the grid ensure that the subset of processors in column $k\ mod\ c$ that owns the diagonal blocks of (block) column k of the matrix, will own the diagonal blocks of any other column that maps to column $k\ mod\ c$.

5 Final Remarks

The documentation on PLAPACK is misleading in places. It is often the case that the advantages of PBMD were contrasted with the disadvantages of a block distribution (a distribution where only one block of the matrix is assigned to any given processor) thereby obscuring the fact that these advantages are enjoyed by all block-cyclic distributions ([2], pp 6-7). However, we credit the PLAPACK authors for talking more about the matrix generation process and for their concern about the libraries-applications interfacing problem. This particular issue has been one that was ignored by many library developers.

References

[1] Almadena Chtchelkanova, Carter Edwards, John Gunnels, Greg Morrow, James Overfelt, and Robert van de Geijn, *Towards usable and lean parallel linear algebra libraries*, Tech. report, The University of Texas at Austin, May 1996.

[2] Carter Edwards, Po Geng, Abani Patra, and Robert van de Geijn, *Parallel matrix distributions: Have we been doing it all wrong?*, Tech. report, The University of Texas at Austin, October 1996.

[3] Dongarra et. al, *LAPACK working note 93, Installation guide for ScaLAPACK*, Tech. report, The University of Tennesee Knoxville, May 1996.

[4] High Performance FORTRAN Forum, *High Performance Fortran language specification*, Tech. report, Rice University, November 1994.

[5] Robert van de Geijn, *A comprehensive approach to designing parallel linear algebra libraries*, Tech. report, The University of Texas at Austin, March 1996.

[6] Robert van de Geijn et al., *Parallel linear algebra package (PLAPACK) users' guide*, Tech. report, The University of Texas at Austin, August 1996.