Hm 97

\ \

Hm-97

Square Matrix Decompositions—Symmetric, Local, Scattered

G. Fox

13 August 1984

I: Introduction

In our original discussion of matrix multiplication and inversion, we proposed a *square* decomposition. Namely, if we have $N = D^2$ processors, then we decompose an $n \times n$ matrix into N $r \times r$ submatrices—each holding r^2 elements ($rD = n$). This is done in the natural way gotten by viewing the matrix as a two-dimensional domain with n^2 regular grid points in it. This is sketched below and for reasons that will become apparent, we will term this the *local square* decomposition.

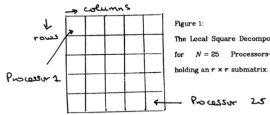


Figure 1:
The Local Square Decomposition for $N = 25$ Processors—each holding an $r \times r$ submatrix.

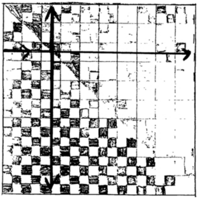
As is apparent, this decomposition naturally uses a two-dimensional grid connection--although the richer interconnect of the hypercube could be important as several algorithms require the "piping" of information between a given processor and all other processors in either the same row and/or the same

Comments

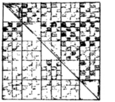
We can now discuss these examples. (a)-(c) are rather similar with (c) being the most elegant. (c) was used in the Jacobi method memo Hm-82.

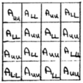
The scattered version of (d)--the *spotted scattered* decomposition is the best "scattered" method as it distributes the load most homogeneously. (e) is unsatisfactory in many algorithms as one often needs a given processor to hold rows/columns of equal length. This allows them to be added/subtracted easily.

One important general point is that processors related by symmetry (i.e. by reflection in diagonal so that $[I,J]$ related to $[J,I]$) are **not** near each other in the two dimensional grid. This causes communication overhead in some parts of the Jacobi algorithm as described in Hm-82. However--one common need--"piping" of information can be handled by "reflecting" pipe when it hits diagonal processor. The example below shows a pipe along the third row (column) of processors.



(d) **The Spotted Symmetric Decomposition**



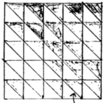

A_{uu} A_{LL} A_{uu} A_{LL}
A_{LL} A_{uu} A_{LL} A_{uu}
A_{uu} A_{LL} A_{uu} A_{LL}
A_{LL} A_{uu} A_{LL} A_{uu}

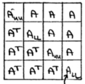
Local Decomposition

Scattered Decomposition

20 × 20 Matrix on a 5 × 5 Processor Array

(e) **The Triangular Symmetric Decomposition**



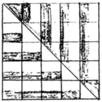


Local Decomposition

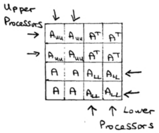
Scattered Decomposition (diagonal assignment depends on choice of storage of elements on "long edge" of triangles in local case)

20 × 20 Matrix on a 5 × 5 Processor Array.

(b) **Another Symmetric Decomposition**



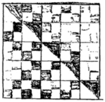
Local Decomposition


Upper Processors
A_{uu} A_{uu} A^T A^T
A_{uu} A_{uu} A^T A^T
A A A_{LL} A_{LL}
A A A_{LL} A_{LL}
Lower Processors

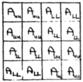
Scattered Decomposition

20 × 20 Matrix on 5 × 5 Processor Array

(c) **The Checkerboard Symmetric Decomposition**



Local Decomposition


A_{uu} A_{uu} A_{LL} A_{LL}
A_{uu} A_{uu} A_{LL} A_{LL}
A_{LL} A_{LL} A_{uu} A_{uu}
A_{LL} A_{LL} A_{uu} A_{uu}

Scattered Decomposition

20 × 20 Matrix on a 5 × 5 Processor Array

Representations of a Symmetric Matrix

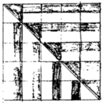
We now need to introduce a little more notation and this is given below.

$$A = A_{LU} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 & 25 \end{bmatrix} \quad A^T = A_{UL} = \begin{bmatrix} 1 & 6 & 11 & 16 & 21 \\ 2 & 7 & 12 & 17 & 22 \\ 3 & 8 & 13 & 18 & 23 \\ 4 & 9 & 14 & 19 & 24 \\ 5 & 10 & 15 & 20 & 25 \end{bmatrix}$$

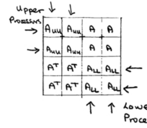
$$A_{UU} = A_{UU}^T = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 7 & 8 & 9 & 10 \\ 3 & 8 & 13 & 14 & 15 \\ 4 & 9 & 14 & 19 & 20 \\ 5 & 10 & 15 & 20 & 25 \end{bmatrix} \quad A_{LL} = A_{LL}^T = \begin{bmatrix} 1 & 6 & 11 & 16 & 21 \\ 6 & 7 & 12 & 17 & 22 \\ 11 & 12 & 13 & 18 & 23 \\ 16 & 17 & 18 & 19 & 24 \\ 21 & 22 & 23 & 24 & 25 \end{bmatrix}$$

In the following we give five examples of different local symmetric decompositions and their *scattered* analog. In each case, the latter is gotten by applying the permutation P of Section II to the former.

(a) **A Possible Symmetric Decomposition**



Local Decomposition



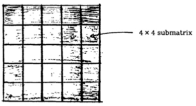
Scattered Decomposition

20 × 20 Matrix on 5 × 5 Processor Array

describe below for a matrix with no symmetry and particular choices $n = 20$ and $r = 4$.

Representations of a General Matrix

The representation of *local* decomposition for a general matrix (no symmetry)



Shaded areas represent elements of a 20 × 20 matrix assigned to a particular processor in a 5 × 5 array.

The corresponding *scattered* decomposition is represented as

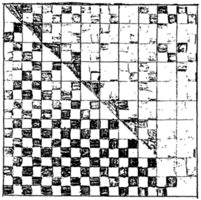
A	A	A	A
A	A	A	A
A	A	A	A
A	A	A	A

Here A represents a 5 × 5 Array of Processor numbers. Expanding the above diagram gives one a 20 × 20 matrix. The number in each location represents the label of the processor holding the matrix element in this location.

used to decompose a four dimensional problem on a two dimensional grid. This was used in the applications of the ICL DAP to QCD by the Edinburgh group.

III: Symmetric Decompositions

We have already discussed the decomposition of symmetric matrices in Hm-82 in the context of the Jacobi eigenvalue technique. There we proposed what we now call the *checkerboard local* decomposition. This is exemplified below for a 10 × 10 processor array. In this diagram we show a matrix divided into a 10 × 10 grid. The shaded areas represent the independent matrix elements assigned to the particular processor. Due to the symmetry, each processor is assigned half (slightly more for diagonal processors) the load used for general matrices.



We need to understand the analogue of the *scattered* format for symmetric matrices. It seems useful to introduce a graphical representation which we first

Clearly the *local* and *scattered* decompositions are just related by applying the permutation P.

Now let us study these decompositions geometrically. The *local* decomposition divides the domain into a $D \times D$ array each containing r^2 matrix elements The *scattered* decomposition divides the domain into a $r \times r$ array each containing $N = D^2$ matrix elements. In the *local* case, one array member is assigned to a single processor. In the *scattered* case, one array member contains one and only one matrix element for each processor.

Note that even in the *scattered* case, each processor holds r columns of r rows; the difference from the *local* decomposition is that the rows (and columns) are not adjacent as they are in the *local* case.

The *scattered* decomposition does not appear to be **needed** in all matrix algorithms; however, it may be **usable** in all. The *local* decomposition is only usable in algorithms without elimination (such as multiplication and Jacobi's eigenvalue technique). I don't know an algorithm where the *local* method is preferable to the *scattered*.

In the matrix case there is no "metric" favoring locality. One needs the two dimensional processor grid to implement the row/column nature of algorithms. However, the ordering within rows/columns is generally irrelevant. For this reason, the *scattered* decomposition with its natural load balancing in systematic algorithms is preferable. There may be other circumstances where the ideas underlying the *scattered* decomposition may be appropriate. We have in mind a two dimensional inhomogeneous problem where we apply the above with each "matrix element" being a subregion of the space. One trades communication overhead (neighboring subregions are no longer adjacent) against load imbalance. Layout of circuit boards could use a *scattered* decomposition. There is also some similarity between the *scattered* decomposition and the techniques

Then any row, k, may be labeled by the pair (I,i) by

$$k = Ir + i + 1$$

where k is i'th row in I'th row of processors for the *local square* decomposition.

Then the permutation P is defined by

$$P_{(I,i)} = iD + I + 1$$

or, equivalently, we have isomorphism

$$Ir + i + 1 \rightarrow iD + I + 1$$

This indicates a nice duality about the transformation.

Now there is an alternative way of viewing this which leads to the *scattered* decomposition. Above we kept the same (*local*) decomposition but changed the algorithm. We can get the same result by applying the old algorithm to a new decomposition.

We define the *scattered square* decomposition by first dividing processors into a square grid. Each processor is labeled by the pair of integers $[I,J]$ with $0 \le I,J \le D - 1$. Then the $[I,J]$'th processor is assigned

$$\begin{array}{lll} \text{rows} & iD + I + 1 & 0 \le i \le r - 1 \\ \text{columns} & jD + J + 1 & 0 \le j \le r - 1. \end{array}$$

This should be contrasted with *local square* decomposition where $[I,J]$ holds

$$\begin{array}{lll} \text{rows} & Ir + i + 1 & 0 \le i \le r - 1 \\ \text{columns} & Jr + j + 1 & 0 \le j \le r - 1. \end{array}$$

Major Step 1: as above

Major Step 2: Eliminate variable x_{r+1} (column $r+1$) from equation $r+1$ (row $r+1$).

Major Step 3: Eliminate variable x_{2r+1} (column $2r+1$) from equation $2r+1$ (row $2r+1$).

and so on.

For this new algorithm, the number of active rows and columns at most differs by 1 between processors whereas in the old algorithm the active rows/columns differed by r between processors. In the old algorithms the inefficiency due to load imbalance was a nonzero constant (independent of r) whereas in the new algorithm the inefficiency is proportional to $1/r$. Both communication overhead and load imbalance disappear as $r \to \infty$ for fixed N.

We can view the algorithm as eliminating row and column P_k in the k'th major step. Here P_k ($k=1..n$) is a permutation of the integers $1..n$ defined by:

$P_1 = 1$

$P_2 = r+1...$

$P_3 = 2r+1$

$P_D = (D-1)r+1$

$P_{D+1} = 2$

and so on.

Formally we define the permutation P as follows. Let capital letters $I,J = 0..D-1$ label rows and columns of processors; let lower case letters $i,j = 0..r-1$ label rows/columns within processors.

column.

Even in the original discussion of inversion it was noted that this *local square* decomposition was not optimal because as one eliminated rows the processors holding them become idle. The same problem also occurs in banded matrix algorithms with pivoting and in the Householder tridiagonalization algorithm. We proposed to "cure" this problem by permuting rows and columns. In Section II, we formalize a particularly useful permutation which leads to the *scattered square* decomposition. Note that both decompositions are usable on a given problem; in systematic elimination algorithms, the *scattered* decomposition has a better load balancing efficiency than the *local* decomposition.

II: Scattered Square Decomposition

This can be motivated by discussing the use of the decomposition in Figure 1 for matrix inversion. The first processor holds columns $1..r$ of rows $1..r$. The first five processors hold together all columns of rows $1..r$ The straightforward Gaussian Elimination (LU decomposition) technique has n major steps--after the $(k-1)$'th such step, rows $1...(k-1)$ are complete and have no further action associated with them. In step k, the activity per row is roughly equal for rows k to n while as above there is no calculation associated with rows $1..k-1$. Similar statements hold for the columns. It follows that with the *local square* decomposition that after r major steps, both the first row and columns of processors (i.e. processors 1,2,3,4,5,6,11,16,21) are idle. This leads to a load imbalance and a reduction of the efficiency of the concurrent algorithm.

The solution of this problem was already described in Hm-5 Namely in the usual algorithm one proceeds as follows: