TR/09/91 **July1991**

THE SCHEDULING OF SPARSE MATRIX-VECTOR MULTIPLICATION ON A MASSIVELY PARALLEL DAP COMPUTER

J. Andersen, G. Mitra, D. Parkinson

Contents

0. Abstract

An efficient data structure is presented which supports general unstructured sparse matrix-vector multiplications on a Distributed Array of Processors (DAP). This approach seeks to reduce the inter-processor data movements and organises the operations in batches of massively parallel steps by a heuristic scheduling procedure performed on the host computer.

The resulting data structure is of particular relevance to iterative schemes for solving linear systems. Performance results for matrices taken from well known Linear Programming (LP) test problems are presented and analysed.

1. Introduction

One of the prerequisites for sparse matrix techniques on a Single Instruction Multiple Data (SIMD) computer is the design of data structures which supports sparse matrix-vector multiplication. Sparse techniques are often indirect in nature and usually handled by list structures which are not very suitable on an SIMD machine. For the parallel machine architectures the efficient design of data structure has reached a new importance. The problem of efficient multiplication of a sparse matrix by a vector on a vector computer has recently been addressed by Erhel [ERHEL90] using a CRAY-2. The DAP is potentially yery powerful when working on dense problems, it is therefore tempting to ignore sparsity and thus saving the overhead of complicated housekeeping logic. This approach might well give good results for moderately sparse matrices or for problems where it is possible to reorder the matrix into a block sparse form.

Our particular interest in generalised sparsity stems from our concern with the parallel computation of the main algorithmic step of the Interior Point Method (IPM) for solving large scale Linear Programming (LP) problems, Karmarkar [KARMAR84], Lustig et al. [LUSMSH89], Andersen et. al. [ANLEVM90].

The main computational work in each step of the IPM amounts to solving a symmetric positive definite system of linear equations. For the SIMD machine architecture we have focused on an iterative Conjugate Gradient solver, as this method can be reduced to repeated matrix-vector multiplication steps which are intrinsically parallel.

Adopting the massively parallel paradigm is a new challenge, it can often lead to novel computational schemes such that apparent restrictions of the SIMD architecture are removed. See also [PARKIN89] .

In this paper we examine the performance of some possible data structures and associated matrix vector multiplication schemes for representing unstructured sparse matrices on the massively parallel architecture of the DAP. In Section 2 we give a brief background to the problem of sparse representation and a few past approaches to this problem. Necessary conceptual tools are introduced in Section 3, whereby dense matrix operations with efficient parallel performance can be represented. The

theoretical algebraic workload which serves as a guidance for comparing the schemes is introduced. These representations form the basis for the further development of sparse schemes, which are described in depth in Section 4. In this section we consider a class of schemes based on block partitioning of the matrix. A particular matrix "STAIR" (Fig. 8.1 in appendix B) which is of the order 356 by 356, is used as a standard example to illustrate differences between the various schemes.

Extensive test results are presented in Section 5 together with implications for special matrices. The analysis of complexity apects as well as concluding discussions are contained in Section 6.

2. Background and Overview

The DAP computer is organised in a 2-dimensional grid of p by p processors; 32 by 32 for the DAP 500 series or 64 by 64 for the DAP 600 series. The memory is distributed such that each processor has some private memory. All the processors are controlled by a single master unit which issues the same instruction to all the processors. The results of a global single instruction, however, can be masked out such that it has no effect for a particular processor, thereby providing some form of local control.

The 2-dimensional grid also defines a fixed communication pattern of rows and column along which the inter-processor communication is most effective. Programs can be written in FORTRAN PLUS; an extension to the FORTRAN language which includes parallel objects, [AMTFOR90] . The relevant language extensions which are referred to in the paper, are itemised in appendix A.

Since the emergence of massively parallel computers, there has been a continued effort in developing numerical software for general purpose large scale computations. The problem of representing sparse matrices on the DAP was first considered by PARKINSON [PARKINS81], he developed his long-vector method which was based on low level permutation code to achieve GATHER/SCATTER type of operations. A long-vector derives from the 1-dimensional indexing of p by p locations forming a general parallel object. While the storage used is compact, there are some inevitable communication overhead involved depending on the regularity of the sparsity pattern. BARLOW, EVANS and SHANECHI, [BAREVS84] proposed an efficient scheme for structured sparse matrix-vector multiplication. Their scheme used PARKINSON'S long-vector method as a kernel, but the sparsity was restricted to banded matrices, these matrices often occur by the discretisation of systems of partial differential equations.

MORJARIA and MAKINSON [MORMAK84] subsequently proposed a general matrix-vector multiplication scheme for the DAP. The attraction of their method is that they eliminated one source of communication between processors. They specified a general stack of DAP memory planes for storing overlaid blocks of the matrix such that there is a fixed relationship between the indices of the element and

its position on the DAP grid. See Section 3. In this way a sparse matrix with unstructured sparseness pattern can be stored compactly on the DAP.

A basic criticism of this method stems from the scattering of blocks of the matrix over the data stack of memory planes. If elements originating from different blocks are stored in the same memory plane in the DAP, then this effectively reduces the inherent parallelism. While the data stack is compact compared with the dense problem, this data structure has to be traversed many times as the elements from matching blocks could be widely scattered in the stack. While this may be manageable for a limited problem scale, there is in fact a complexity argument against the method; this is discussed in Section 6. As the matrix size grows, so does the data stack, as a consequence, the number of references necessary for the parallel computations multiplies the computational complexity measure by the problem size, thus eventually defeating the parallelism. On the positive side however, the use of block indices simplifies the housekeeping and accordingly provides for a simple transition to dense matrix mode with full parallel performance.

3. Basic concepts and definitions

3.1 Block partitioning methods

Consider a matrix A of size MxN partitioned into a number of DAP size (p by p) block matrices $B^{k,\ell}$ such that:

$$[A]_{i,j} = [B^{k\,\ell,l}]_{m,n} \qquad (3.1)$$

$$i = 1,..,M \quad j = 1,...,N \text{ and } m,n = l,...,p$$

In (3.1) we define (i,j) as global indices for A, (k, ℓ) as block indices and (m,n) as local indices, see also Fig. 3.1 .

The correspondence between the indices is given by the relations:

global indices

$$\begin{cases} i = p\,(k-1) + m \\ j = p(f-1) + n \end{cases}$$

with k= 1,..., (M- l)/p+ 1

and $\ell = 1\ ,\ \ldots\ ,\ (N-1)/p+1$

(truncated integer division)

The inverse correspondence relations are:

block indices.

$$\begin{cases} k = 1 + (i-1) \bmod p \\ \ell = 1 + (j-1) \bmod p \end{cases}$$

local indices

$$\begin{cases} m = k - (i-1)\,p \\ n = \ell - (j-1)\,p \end{cases}$$

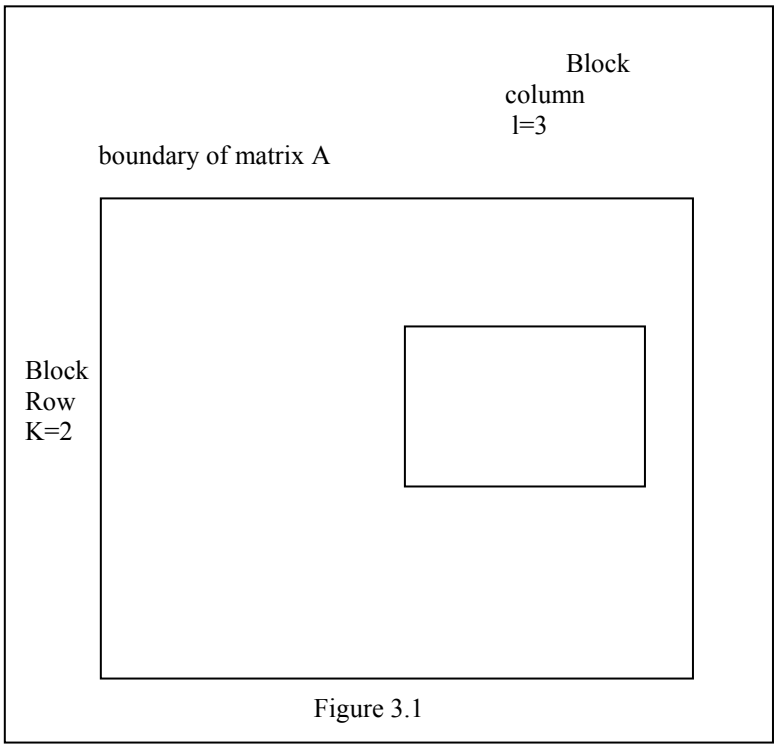


Figure 3.1

We introduce the term: "block row (column) index" for labelling the blocks. To be concise, a row (column)-block is a special kind of block which has a dimension of pxN (Mxp), hence a "row (column)-block index" is a label for these rectangular structures.

For the block structure of the matrix, we define:

$$\text{MBLK} = (M\text{-}1)/p+1 \text{ , and}$$

$$\text{NBLK}=(N\text{-}1)/p+1$$

We wish to compute the sparse matrix-vector multiplication in the assignment

$$y = Ax \qquad (3.2)$$

This can be stated in terms of the block specific components as:

$$y^k \sum_{\ell=1}^{\text{NBLK}} B^{k,\ell} X^{\ell}, \quad k = 1,...,\text{MBLK} \qquad (3.3)$$

Here x^{ℓ} and y^{k} refers to a p element vector block of the x and y vectors respectively.

For the analysis of the potential parallelism, we define a special multiplication operator $\otimes$ for the direct multiplication of parallel objects. If two block matrices are stored as parallel objects, then the $\otimes$ operation between them corresponds to component wise multiplication. In order to exploit the parallelism we rewrite (3.3) in an outer product form, using the parallel $\otimes$ operations between block matrices, to do this it is convenient first to generate a temporary block matrix by copying x^{ℓ}; the ℓ'th block of the x-vector onto all the rows of a temporary block matrix. This broadcasting operation can be expressed symbolically as:

$$X^{k,\ell} = ex^{\ell(\text{Transposed})}, \qquad \text{Where } e = \begin{bmatrix} 1 \\ . \\ . \\ . \\ 1 \end{bmatrix}$$

$X^{k,\ell}$ becomes a (p by p) block matrix. With these definitions, we can represent the m'th component of y^k in terms of the parallel objects:

$$y_m^k = \sum_{\ell=1}^{NBLK} \sum_{n=1}^{P} [B^{k,\ell} \otimes X^{k,\ell}]_{m,n} \qquad (3.4)$$

By defining $Y^{k,\ell} = B^{k,\ell} \otimes X^{k,\ell}$, equation (3.4) becomes:

$$y_m = \sum_{\ell=1}^{NBLK} \sum_{n=1}^{P} [Y^{k,\ell}]_{m,n} \qquad (3.5)$$

The summation which is required inside the block is not fully parallel, it requires approximately $\log_2$ (p) parallel steps using the well known parallel cascade summation. The summation however need not be carried out before a complete row block is ready; instead of NBLK individual cascade sums, only one final summation is necessary for each row block. The result for a row block is:

$$y_n^k = \sum_{n=1}^{P} \sum_{\ell=1}^{NBLK} \left[Y^{k.\ell}\right]_{mn} \qquad (3.6)$$

The last result is obtained by changing the summation order in (3.3), it follows that the inner summation can be evaluated in (massively) parallel steps, leaving just one cascade sum for each row block A build in function "sumc" is used for maximum efficiency for this task.

For iterative applications, the x-vector is constantly modified while the A matrix is unchanged. Therefore any sparsity pattern in the x-vector is not useful unless it has a void block(s). The testing for a zero void is again a massively parallel operation which can be performed as a single Boolean step.

By using such primitives, a dense matrix-vector multiplication can be conveniently represented. For the analysis of the computational workload involved in the algorithms, it is also useful to define two parallel measures:

DFLM = number of DAP size floating point multiplications,

similarly:

DFLA = number of DAP size floating point additions.

These definitions are intended only to clarify the analysis of parallel performance.

3.2 Dense mode algorithms

The parallel workload for the dense mode algorithm is:

$$DFLM(dense) = MBLK*NBLK \quad (3.7)$$

The required number of parallel additions however, depends critically on the order of summation in the above expressions or whether equation (3.5) or (3.6) is used. With standard functions available such as "sumc", the immediate application of the summation as in (3.5) leads to the measure:

$$DFLA*(dense) = (NBLK - 1)*\log_2(p)*MBLK \quad (3.8)$$

While using the postponed summation as in (3.6) gives the superior measure: D

$$DFLA(dense) = (NBLK - 1 + \log_2 (p))*MBLK \quad (3.9)$$

As an example, we calculate these measures for a matrix which has the dimension 356x356. Assuming 12 by 12 blocks and p = 32 we find;

DFLM(dense)	DFLA*(dense)	DFLA(dense)
144	660	192

3.3 The block sparse mode

The block sparse mode is a natural extension to the parallel dense mode scheme as outlined in 3.2. In this method any void block which matches the block partition of the matrix is bypassed if detected. The logical testing of a parallel object for zero entries can be done as a single massively parallel step in one machine cycle only, hence it represents virtually no overhead. Alternatively, by analysing the pattern in advance of the computations, the stack of memory planes required for the data structure might be reduced.

Unless the matrix exhibits a clear structured pattern, there are few chances of such void blocks occurring, even if the matrix can be treated as sparse on a serial computer. There is, however, scope for changing this situation by reordering the matrix using a strategy which seeks to concentrate the nonzeros into islands leaving some clear patches.

The well known Cuthill-McKee matrix ordering scheme for symmetric matrices

seeks to concentrate the matrix elements towards the diagonal (minimum bandwidth ordering), a program for the algorithm is described by George and Liu [GEOLIU81].

4. Sparse algorithms

A general class of sparse storage schemes consist of overlaying the matrix blocks where possible, while keeping track of the block column and block row indices for the local occupants of the blocks in a matching data structure of indices.

The time taken for setting up the data structure can be justified if the repeated matrix-vector multiplications is required on the DAP, using the same matrix in each iteration. An example is the Conjugate Gradient linear equation solver.

In the following it is useful to refine the terminology such that the overlaid blocks in the data structure is referred to as "planes" as the structure has some characteristics of a stack. The term "blocks" hereinafter indicates the distinct matrix blocks before any data packing has taken place.

4.1 MORJARIA and MAKINSON's (MM) method

In the MM method, the matrix element $[A]_{ij}$ is assigned to processor (m,n) as specified by equation 3.1 . This mapping causes many elements to be allocated to the same processor so a stack of elements is created in each processor. An exampl of a multiple stack, which follows from this data mapping is presented in Fig. 4.1. For our standard test matrix "STAIR" (see section 5.1), the maximum number of elements allocated to any processor by the MM method is 22 which is very much less than the 144 which would be the maximum if the matrix is treated as dense. The actual depth of the stack is different for each processor. We define a "plane" as the set of elements found at a given depth in the processors, thus elements in a plane are potential candidates for parallel operations.

If we examine a given plane in the multiple stack, (see Table 4.1) we find that the elements in that plane derive from different (k, ℓ) blocks. For example the plane at the fifth level of the data structure derived for STAIR has 9 different k-values and 11 different ℓ-values. The number of "broadcast operations" of the x-vector to deal with a given plane is the same as the number of different column blocks represented within that plane. If the multiplications are done immediately after each broadcast, many multiplications are needed (157) for the STAIR matrix. If, however, the result of the broadcasts are assigned under masking to a common temporary variable (merging) then all the multiplications for a given plane may be performed simultaneously, reducing the total number of multiplications to the maximum stack depth. It is not clear from Morjaria and Makinson's paper [MORMAK84] that they made this observation. As broadcasting on the DAP is very much faster than multiplication the operation time is very much reduced. The total number of parallel m multiplications is given by the maximum stack depth (22 in case of STAIR).

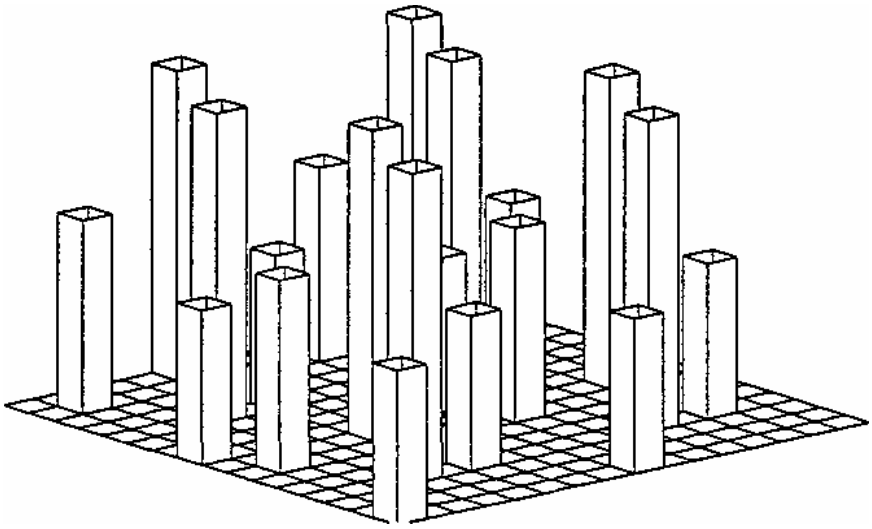


Fig. 4.1 A multiple stack on p by p processors

To perform the summation operations is more difficult as each row block in a plane must be dealt with separately, so the number of parallel additions is strongly dependent on the distribution of block indices. A detailed map of the data structure is given in Table 4.1; each symbol "X" indicates that one or more matrix elements of the same row block is stored in a particular plane. It is apparent that elements from each row block is scattered over many planes, totalling 156 when adding the number of planes used by all 12 block row indices. The DFLA measure for the addition step can be found by the same approach as was outlined in section 3.2 equation 3.9; each row block results in an additional cascade summation requiring $\log_2(32)$ parallel operations.

$$DFLA(MM) = 156 - 12 + \log_2(32) * 12 \qquad (3.10)$$

DFLM*(MM)	DFLM(MM)	DFLA(MM)
157	22	204

Table 4.1

Morjaria/Makinson (MM) method for matrix: STAIR

Plane	Block row indices (k) present present in planes												Sum
	1	2	3	4	5	6	7	8	9	10	11	12	
1	X	X	X	X									4
2	X	X	X	X	X	X							6
3	X	X	X	X	X	X	X	X	X				9
4	X	X	X	X	X	X	X	X	X	X			10
5	--X--	X--	X--	X--	X--	X--	X--	X--	X-	----------			9
6		X	X	X	X	X	X	X	X	X	X		10
7		X	X	X	X	X	X	X	X	X	X		10
8		X	X	X	X	X	X	X	X	X	X		10
9				X	X	X	X	X	X	X	X		8
10				X	X	X	X	X	X	X	X		8
11				X	X	X	X	X	X	X	X	X	9
12					X	X	X	X	X	X	X	X	8
13						X	X	X	X	X	X	X	7
14						X	X	X	X	X	X	X	7
15						X	X	X	X	X	X	X	7
16						X	X	X	X	X	X	X	7
17						X	X	X	X	X	X		6
18							X	X	X	X	X	X	6
19							X	X	X	X	X		5
20							X	X	X	X	X		5
21									X	X	X		3
22											X	X	2
Totals	5	8	8	11	11	16	18	18	19	17	17	8	156
-1+log	4	4	4	4	4	4	4	4	4	4	4	4	48
DFLA	9	12	12	15	15	20	22	22	23	21	21	12	204

Plane	Block column indices (l) present in planes												Sum
	1	2	3	4	5	6	7	8	9	10	11	12	
1	X	X	X	X	X	X							6
2	X	X	X	X		X							5
3	X	X	X	X	X	X		X	X	X			9
4	X	X	X	X	X	X	X	X	X				9
5	-- X--	X--	X--	X--	X--	X--	X--	X--	X--	X--	X	---	11
6		X	X	X	X	X	X	X	X	X			9
7		X	X	X	X	X	X	X	X	X	X		10
8		X	X	X	X	X	X	X	X	X	X		10
9				X	X	X	X	X	X	X	X		8
10				X	X	X	X	X	X	X	X		8
11				X	X	X	X	X	X	X	X	X	9
12					X	X	X	X	X	X	X	X	8
13						X	X	X	X	X	X	X	7
14						X	X	X	X	X	X	X	7
15						X	X	X	X	X	X	X	7
16						X	X	X	X	X	X	X	7
17						X	X	X	X	X	X		6
18							X	X	X	X	X	X	6
19							X	X	X	X	X		5
20							X	X	X	X	X		5
21									X	X	X		3
22											X	X	2
Totals	5	8	8	11	11	17	17	18	19	18	17		157

Potential problems with the scheme can occur when the matrix has a semi regular pattern **modulo(p)** which can cause large stack sizes.

4.2 The Extended stacking scheme (ESS)

While the MM scheme usually has a compact data structure, the number of parallel additions required as shown indicate a poor exploitation of sparsity in this respect. In the extended stacking scheme (ESS) we separate out the different row block indices and for each index set up a local stack in each processor as with the global stack for the MM scheme.

Table 4.2 shows the presence of block row and block column indices in each plane of the data structure. A preliminary scan over the matrix elements determines the number of planes required for each local stack. These local stacks can be envisaged as sealed boxes, which are again stacked, though the boxes do not overlap. Because the overlap is inhibited, the data structure is less compact than for the MM scheme. For the example STAIR the total depth of this stack of stacks is 43 instead of 22 as in the MM scheme. On the other hand, the matching data structure to indicate the block row index of each element stored is no longer needed.

By separating out the row blocks in this way, the number of parallel additions for the STAIR matrix becomes $43+(\log_2(p)-1)*MBLK$ or 91, while the number of multiplications can be limited to 43 by using the masked broadcasting technique as previously described for the MM method. For the price of 21 extra multiplications we achieved a saving of 113 parallel additions.

DFLM(ESS)	DFLA(ESS)
43	91

With this scheme, a merge operation of the broadcasted x-vector takes place for each block column index present in a given plane. For the STAIR example the merging of up to 12 broadcasts are required before each parallel multiplication. While the merging of broadcasts is much cheaper than the multiplication on the DAP, it is still possible to assign the matrix elements in the local stack in such a way that the number of different block column indexes in a plane is kept as low as possible. Such a strategy is particularly beneficial if the DAP includes special fast floating point processing elements, this DAP is also known as the coprocessor version. A special heuristics for assigning the matrix elements such that the number of broadcasts is reduced is as follows.

Table 4.2

"Extended stack" (ESS) allocation method for STAIR

Plane	Block row indices (k) present in planes												Sum
	1	2	3	4	5	6	7	8	9	10	11	12	
1	X												1
2	X												1
3	X												1
4		X											1
5		X											1
6		X											1
7			X										1
8			X										1
9			X										1
10			X										1
11				X									1
12				X									1
13				X									1
14				X									1
15					X								1
16					X								1
17					X								1
18					X								1
19						X							1
20						X							1
21						X							1
22						X							1
23						X							1
24							X						1
25							X						1
26							X						1
27							X						1
28								X					1
29								X					1
30								X					1
31								X					1
32								X					1
33									X				1
34									X				1
35									X				1
36										X			1
37										X			1
38										X			1
39										X			1
40											X		1
41											X		1
42											X		1
43												X	1
Totals	3	3	4	4	4	5	4	5	3	4	3	1	43
-1+log	4	4	4	4	4	4	4	4	4	4	4	4	48
DFLA	7	7	8	8	8	9	8	9	7	8	7	5	91

Plane	Block column indices (l) present in planes												Sum
	1	2	3	4	5	6	7	8	9	10	11	12	
1	X	X	X										3
2		X	X										2
3			X										1
4	X	X	X	X									4
5		X	X	X	X								4
6			X	X	X								3
7	X	X	X	X	X	X							6
8		X	X	X	X								4
9			X	X	X								3
10					X								1
11		X	X	X	X	X							5
12			X	X	X	X	X						5
13				X	X	X							3
14						X							1
15		X	X	X	X	X	X						6
16			X	X	X	X	X						5
17				X	X	X							3
18						X							1
19			X	X	X	X	X	X					6
20					X	X	X	X					4
21						X	X	X					3
22							X	X					2
23								X					1
24				X	X	X	X	X	X				6
25					X	X	X	X	X	X			6
26						X		X	X	X			4
27							X						1
28						X	X	X	X	X	X		6
29							X	X	X	X			4
30								X	X	X			3
31									X	X			2
32										X			1
33							X	X	X	X	X		5
34								X	X	X	X		4
35										X	X		2
36							X	X	X	X	X	X	6
37								X	X	X	X	X	5
38									X	X	X		3
39										X	X		2
40								X	X	X	X	X	5
41										X	X	X	3
42											X		1
43											X	X	2
Totals	3	8	14	14	16	16	14	16	13	17	12	4	147

Table 4.3

"Extended stack" (ESS) allocation method with priority assignments for STAIR

Plane	Block row indices (k) present in planes												Sum
	1	2	3	4	5	6	7	8	9	10	11	12	
1	X												1
2	X												1
3	X												1
4		X											1
5		X											1
6		X											1
7			X										1
8			X										1
9			X										1
10			X										1
11				X									1
12				X									1
13				X									1
14				X									1
15					X								1
16					X								1
17					X								1
18					X								1
19						X							1
20						X							1
21						X							1
22						X							1
23						X							1
24							X						1
25							X						1
26							X						1
27							X						1
28								X					1
29								X					1
30								X					1
31								X					1
32								X					1
33									X				1
34									X				1
35									X				1
36										X			1
37										X			1
38										X			1
39										X			1
40											X		1
41											X		1
42											X		1
43												X	1
Totals	3	3	4	4	4	5	4	5	3	4	3	1	43
-1+log	4	4	4	4	4	4	4	4	4	4	4	4	48
DFLA	7	7	8	8	8	9	8	9	7	8	7	5	91

Plane	Block column indices (l) present in planes												Sum
	1	2	3	4	5	6	7	8	9	10	11	12	
1	X												1
2		X											1
3			X										1
4	X			X	X								3
5		X		X	X								3
6			X										1
7	X				X								2
8		X				X							2
9			X										1
10				X	X								2
11		X				X							2
12			X				X						2
13				X									1
14					X	X							2
15		X					X						2
16			X			X							2
17				X									1
18					X	X							2
19			X					X					2
20				X									1
21					X								1
22							X						1
23						X							1
24				X				X					2
25					X				X				2
26						X		X					2
27							X			X			2
28						X					X		2
29							X						1
30								X					1
31									X				1
32										X			1
33							X			X	X		3
34								X		X			2
35									X	X			2
36							X				X		2
37								X			X		2
38									X			X	2
39										X	X		2
40								X			X		2
41									X			X	2
42										X			1
43										X	X		2
Totals	3	5	6	7	8	8	7	7	5	8	7	2	73

Using the preliminary scan, as above, for determining sizes of the local stacks, an incoming matrix element with a given row block index can be assigned to any plane in the local stack for the block row index where space is free. Instead of assigning the element to the first free location under the processor to which the element belongs, a scan is made to see if there is a free location in the local stack which already has a matrix element of the same column index in the same plane but for a different processor. The scan is only done over a list of column indices for each plane in the local stack, it is referred to as a "matching" scan. If there is no matching location then the assignment is made to a free non-matching location.

A superior assignment technique for both matching and non-matching locations is possible if there is a choice of available locations. In this case the plane in the local stack is chosen which has the smallest number of different block column indices already present. It was found that this strategy reduces the chance of "blocking" of matching locations for subsequent incoming elements. The results of the above enhancements to the ESS scheme from using these "priority assignments" is shown in Table 4.3. The improvement in the allocation of column rows for STAIR in apparent when comparing Table 4.3 with Table 4.2; only 73 merge operations is required instead of 147.

4.3 The Block banded scheme BBS

For the solution of large scale LP problems by using the interior point method (IPM) we are dealing with very sparse matrices where the number of elements in a single row block is often smaller than the number of processors in the massively parallelsystem. To deal with such matrices we introduce a new level of aggregation; the *block band.*

This method generalises the extended stack scheme by allowing matrix elements within a range of row blocks to be allocated to planes of a common stack system. Table 4.4 illustrates the scheme for the STAIR example. The MM scheme can be looked at as an extreme case where the range of block row indices extends to all of them, hence there is only one block band and consequently just one common stack system. Furthermore, the MM scheme does not include any block matching heuristic for the critical assignments of matrix elements to the planes of the stack system.

The width of the block bands are adjusted by a parameter Φ of the scheme as follows: Initially, the block bands are the same as the row blocks, but if successive row blocks has a cumulative count of the nonzero matrix elements which is less than the parameter Φ, then the sequence of row blocks are aggregated into one block band. The ESS scheme can be derived by setting $\Phi = 0$ such that no aggregation of row blocks takes place.

The performance of the scheme thus depends on selecting the parameter Φ to obtain a balance between compactness of the data structure and computational overheads. Table 4.4 illustrate the workings of scheme for the STAIR matrix and selecting $\Phi = 1000$. In this case there is a small further reduction in the total number of DAP size floating point operations.

Table 4.4

"Block banded" (BBS) allocation method for STAIR

aggregation parameter = 1000

Plane	Block row indices (k) present in planes												Sum
	1	2	3	4	5	6	7	8	9	10	11	12	
1	X												1
2	X												1
3	X												1
4		X											1
5		X											1
6		X											1
7			X										1
8			X										1
9			X										1
10			X										1
11				X									1
12					X								1
13				X									1
14					X								1
15				X	X								2
16				X	X								2
17						X							1
18						X	X						2
19						X	X						2
20							X						1
21						X							1
22						X	X						2
23								X					1
24								X					1
25								X					1
26								X					1
27								X					1
28									X				1
29									X				1
30									X				1
31										X			1
32										X			1
33											X		1
34										X			1
35										X			1
36											X		1
37											X	X	2
Totals	3	3	4	4	4	5	4	5	3	4	3	1	43
-1+log	4	4	4	4	4	4	4	4	4	4	4	4	48
DFLA	7	7	8	8	8	9	8	9	7	8	7	5	91

Plane	Block column indices (l) present in planes												Sum
	1	2	3	4	5	6	7	8	9	10	11	12	
1	X												1
2		X											1
3			X										1
4	X			X	X								3
5		X		X	X								3
6			X	X									2
7	X				X								2
8		X				X							2
9			X										1
10				X	X								2
11		X	X		X	X							4
12		X	X		X	X	X						5
13			X	X	X	X	X						5
14			X	X	X		X						4
15				X	X	X							3
16				X	X	X							3
17			X	X	X			X					4
18				X	X		X	X	X	X			6
19					X	X	X		X				4
20					X	X	X	X	X				5
21						X	X	X					3
22						X	X	X	X	X			5
23						X					X		2
24							X						1
25								X					1
26									X				1
27										X			1
28							X			X	X		3
29								X		X			2
30									X	X			2
31							X	X		X	X		4
32								X	X	X	X		4
33								X	X	X			3
34									X	X	X		3
35										X	X	X	3
36										X	X	X	3
37										X	X		2
Totals	3	5	8	10	14	11	11	10	9	13	8	2	104

DFLM(BBS)	DFLA(BBS)
37	91

The block matching algorithm is a feature of both the ESS and the generalised BBS scheme, however the matching of block row indices within a block band becomes an essential part of the BBS scheme for reducing the DFLA count (See also Section 3). The flowchart of the algorithm is set out in Fig. 4.2. For each incoming matrix element the range of planes belonging to the block band of the matrix element is scanned in a priority order which is continuously updated. The priority order is conveniently expressed as the ascending order of values for the function:

$$f(s) = (NBLK+1)*\eta\{K\} + \eta\{L\} \qquad (4.1)$$

Here $\eta\{K\}$ indicate the cardinality of the set of block row indices already present in the plane s and $\eta\{L\}$ is similarly the cardinality of the set of block column indices.

As we have : $\eta[L] \leq NBLK$, the ordering of the function f(s), which is the order in which the planes in the block band is scanned, gives first priority to limiting the number of different block row indices followed by limiting the number of different column block indices. If there is no possible block matching in the block band, then choosing a plane s with the smallest f(s) is a heuristic which seeks to prevent scattering of block indices over many planes.

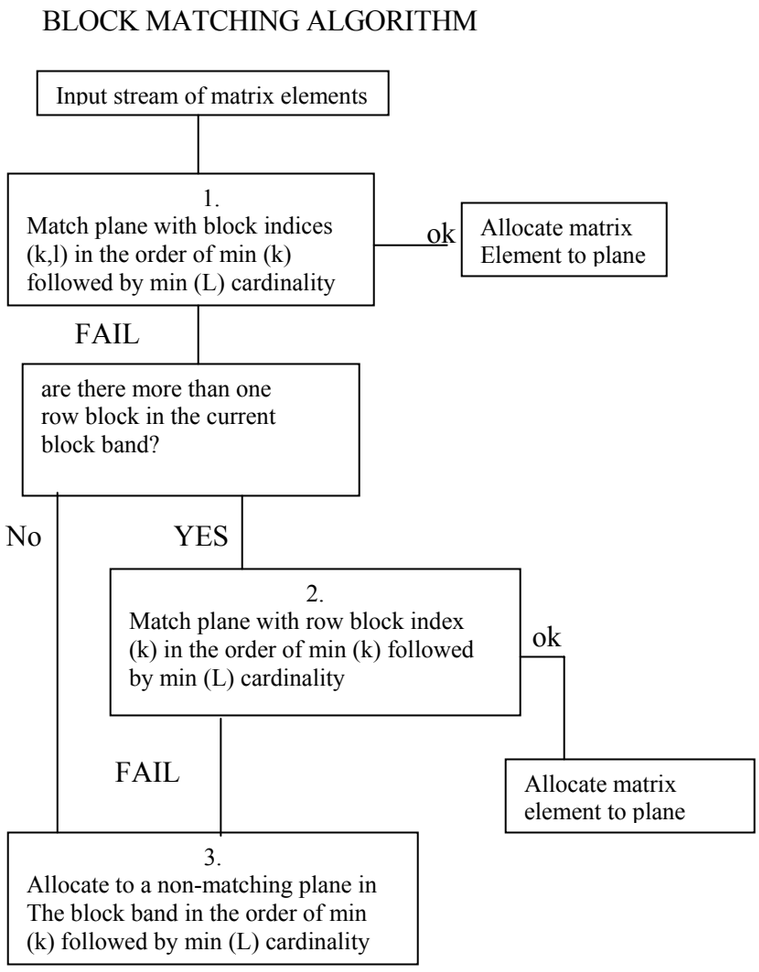

BLOCK MATCHING ALGORITHM
Input stream of matrix elements
1.
Match plane with block indices (k,l) in the order of min (k) followed by min (L) cardinality
ok
Allocate matrix Element to plane
FAIL
are there more than one row block in the current block band?
No
YES
2.
Match plane with row block index (k) in the order of min (k) followed by min (L) cardinality
ok
FAIL
Allocate matrix element to plane
3.
Allocate to a non-matching plane in The block band in the order of min (k) followed by min (L) cardinality

Fig 4.2

5. Presentation and discussion of results

5.1 Database of test problems

The following 8 models have been extracted from well known international benchmarks of LP test problems; "NETLIB" as supplied by Gay [GAYM85] . "Kenn" is from to Kennington et. al. [CARHKW89] and "KLOTZ" refers to a set of problems by courtesy of Klotz [KLOTZ91].

5.1 LP Problems statistics

No	Name	Rows	Columns	Nonzeros	Source
1	STAIR	357	467	3857	NETLIB
2	PILOTXWE	723	2789	9218	NETLIB
3	25FV47	822	1571	11127	NETLIB
4	PILOTXIA	941	1988	14706	NETLIB
5	PILOT	1442	3652	43220	NETLIB
6	CYCLE	2234	2857	31476	KLOTZ
7	CRE_C	3068	3678	13244	KENN
8	CRE_A	3516	4067	14987	KENN

After preprocessing of the LP problems, the total number of nonzeros in the matrix AA^T less the diagonal are set out in Table 5.2.

5.2 Nonzeros in the AA^T matrix less diagonal elements.

No	Name	Rows	Nonzeros	Source
1	STAIR	356	12394	NETLIB
2	PILOTXWE	722	9650	NETLIB
3	25FV47	820	22148	NETLIB
4	PILOTXIA	924	26500	NETLIB
5	PILOT	1441	119080	NETLIB
6	CYCLE	1889	55412	KLOTZ
7	CRE_C	2986	37810	KENN
8	CRE_A	3428	41496	KENN

The sparsity pattern for the matrices in the AA^T form is shown in the figures (Fig. 8.1-8.8 in appendix B). For the display of large matrices each dot represents a submatrix as indicated in the labeling, e.g. "CYCLE 1889/5 57301" indicates that a single dot represents a 5 by 5 submatrix. The adjacent number 57301 is the number of nonzero elements including the diagonal elements which were ignored for the tests.

5.2 Summary results

The test was carried out on a DAP510 with 8 MB of memory. The computer was run both with and without installation of the floating point coprocessor. The maximum possible size of the datastructure with this machine corresponds to 700 planes of 1024 single precision numbers, this excludes the auxcillary data structures necessary for carrying out the matrix vector multiplication.

With respect to the test experiments, the example matrices can be classified into two groups; the smaller matrices (1-4) which fits the memory in dense or block sparse form, and the larger matrices (5-8) which only fits the memory when using a compact (overlaid) scheme. Furthermore, the matrices can be subdivided into block sparse (1,2,6-8) and scattered (3-5).

The MFLOP rate was calculated by counting the minimum number of floating point operations required on the nonzero elements only. The calling of the matrix vector multiplication subroutine was included in the timings. (See Table 5.3-5.4)

The Block sparse scheme BLSPAR performs very well for the group of smaller matrices which can fit the memory without using overlaid packing. Introducing the Cuthill-McKee reordering to the scheme BLSPAR (C-MK) gives only a small improvement for the scattered matrices: 25FV47, PILOTXJA, while this reordering gave no improvement for the block sparse group: STAIR, PILOTXWE.

The extended stacking scheme (ESS) is especially relevant for the larger matrices where it performs about 3 times better than the MM scheme on CRE_A, CRE_C and CYCLE.

The block banded scheme BBS suffer a small overhead compared with the EES scheme because it require an indicator array for the block row indices as well as for the block column indices. This scheme however can be combined with matrix reordering which is indicated by the labelling: n - no reordering, c - Cuthill-McKee reordering. The number in the label is the above mentioned aggregation parameter (see Section 4.3), thus n0 indicates no reordering and no aggregation. The effect of the scheme is most significant for the matrix PILOT which is large and scattered. Only by using the BBS scheme combined with Cutthill-McKee reordering method was it possible to fit the matrix into the memory without any aggregation. This case performed about 6 times better than the unordered MM scheme. By using a aggregation parameter of 4000 the memory requirement for the datastructure can be further reduced from 442 down to 342 planes with only a small cost in speed.

5.3 MFLOP RATE WITHOUT COPROCESSOR

	DENSE	BLSPAR	C-MK	MM	ESS
STAIR	0.53	0.52	0.98	0.72	1.25
PILOTXWE	0.12	0.28	0.22	0.32	0.49
25FV47	0.28	0.31	0.33	0.29	0.64
PILOTXJA	-	0.29	0.29	0.33	0.60
PILOT	-	-	-	-	-
CYCLE	-	-	-	0.28	0.78
CRE_C	-	-	-	0.08	0.33
CRE_A	-	-	-	0.07	0.34

	BBS n0	BBS c0	BBSc1000	BBSc4000
STAIR	1.21	-	-	-
PILOTXWE	0.48	-	-	-
25FV47	0.62	-	-	-
PILOTXJA	0.59	-	-	-
PILOT	-	1.00	1.00	0.96
CYCLE	0.76	0.74	0.75	
CRE_C	0.32	-	-	-
CRE_A	0.33	-	-	-

5.4 MFLOP RATE WITH COPROCESSOR

	DENSE	BLSPAR	C-MK	MM	ESS
STAIR	1.99	2.77	2.73	1.41	2.37
PILOTXWE	0.52	0.89	0.79	0.57	0.82
25FV47	1.00	1.27	1.33	0.61	1.15
PILOTXJA	-	1.21	1.25	0.68	1.03
PILOT	-	-	-	0.29	-
CYCLE	-	-	-	0.59	1.47
CRE_C	-	-	-	0.17	0.54
CRE_A	-	-	-	0.16	0.54

	BBS n0	BBS c0	BBSc1000
STAIR	2.23	-	-
PILOTXWE	0.77	-	-
25FV47	1.07	-	-
PILOTXJA	1.00	-	-
PILOT	-	1.74	1.71
CYCLE	1.36	1.15	-
CRE_C	0.51	-	-
CRE_A	0.51	-	-

5.5 MEMORY PLANES REQUIRED

	DENSE	BLSPAR	C-MK	MM	ESS
STAIR	144	64	67	22	43
PILOTXWE	529	190	252	20	71
25FV47	676	443	407	41	136
PILOTXJA	(841)	596	562	46	162
PILOT	(2116)	-	-	642	-
CYCLE	(3600)	-	-	84	296
CRE_C	(8836)	-	-	115	386
CRE_A	(11664)	-	-	135	394

	BBS n0	BBS c0	BBSc1000	
STAIR	43	-	-	-
PILOTXWE	71	-	-	-
25FV47	136	-	-	-
PILOTXJA	162	-	-	-
PILOT	-	442	425	342
CYCLE	296	247	202	-
CRE_C	386	-	-	-
CRE_A	394	-	-	-

6. Analysis and Conclusions

6.1 Analysis

The performance of the MM scheme for the matrix PILOT gives an indication of the complexity aspect of this method. If the matrix is treated as dense it would require the following number of DAP size operations:

DFLM(dense)	DFLA(dense)
2116	2300

While analysing the data stack for the MM scheme for PILOT gives the measures:

DFLM(dense)	DFLA(dense)
642	11350

The resulting DFLA count (11350) is 38% of the maximum possible figure if the whole data structure of 642 planes had to be traversed for each row block. This illustrates the point (Section 2.) that in the MM scheme, a large fraction of the data structure has to be traversed for summation operation in each row block.

The development of sparse matrix techniques has made it possible to solve large scale LP problems on computers of moderate size, although even for serial computers, there are some considerable overhead involved with the indirect addressing of the data structure. The salient features of the proposed data structure, which supports sparse matrix vector multiplication on the massively parallel DAP computer are the overlaid blocks and the block matching heuristic.

Generally, the use of overlaid blocks is advantageous for reducing the communication between processors. The block matching technique also contributes in reducing the communication and is a key to parallelism for the more compact BBS scheme. The figure (Fig 6.1) shows an example of the number of DAP size floating point operations required for the matrix PILOTXJA as a function of the aggregation parameter Φ , the curve marked "DFL" shows the combined number of DAP size floating point operations, this curve is almost flat over the range of F starting with Φ=0 corresponding to MBLK block bands and up to Φ =26500 which corresponds to just one block band. However, with increasing aggregation more communication is likely to follow, this is evidenced by the presence of block column indices in the data structure shown in Table 4.4 in comparison with Table 4.3 .

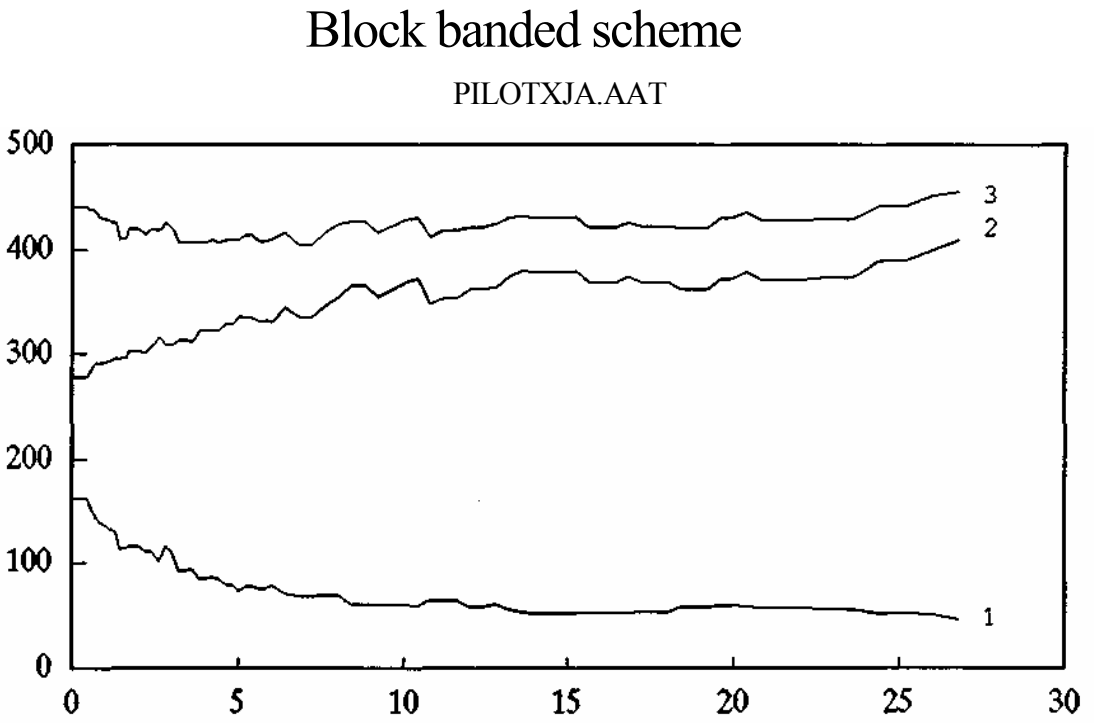


Fig. 6.1

6.2 Conclusions

The two main issues with the proposed scheme are the folllowing

1) Inefficient packing of overlaid blocks (collisions)
2) Non-matching of block indices in the planes (scattering)

The first issue arises if there is a regular pattern in the matrix which matches the block size; if this is caused by some form of structured sparsity, then this could be exploited in different ways. If the only apparent structure is a repeated block pattern, then the collision problem could be resolved by mapping a slightly different blocksize e.g. (p-1) by (p-1) onto the p by p processor grid. A single dense row or column can also lead to inefficient packing, but if detected the contribution from such rows or colums to the matrix-vector product can be computed separately by highly parallel methods. The matrices CRE_A and CRE_C contains dense rows and columns which explains the reduction of performance in these cases. However, no provision was made for the separate treatment of the dense rows and columns.

In dealing with the second issue we would like to use a re-ordering method which improves the block sparsity pattern and thus simplifying the block matching procedure. The Cuthill-McKee re-ordering was only beneficial for the scattered matrices: 24FV47, PILOTXJA and PILOT, while the re-ordering did not improve the performance for the matrices STAIR , PILOTXWE and CYCLE which were already block sparse.

7. Acknowledgements

Mr. Johannes Andersen is supported by the Science and Engineering Research Council (UK) CASE studentship with Active Memory Technology (AMT) as the industrial sponsor. The authors gratefully acknowledge the equipment facilities offered by AMT and Queen Mary & Westfield College.

8. Appendix

Apendix A

The FORTRAN PLUS language contains some parallel extensions to FORTRAN77. The '*' in front of the first array dimension number in some of the array dimension statements indicate that the variable is a parallel object. The new FORTRAN-PLUS 'enhanced' will map the objects onto the machine architecture as parallel as possible, the compiler is therefore also called "unrestricted", this greatly facilitates portability between DAP computers of different size, e.g. from the 32 by 32 grid array of processors of the AMT DAP510 to the DAP610 which has a grid of 64 by 64 processors.

If an array is declared with a parallel dimension, then the index can be omitted in a statement with the implied meaning "for all elements" of the parallel dimension. An operation can also be selectively masked by substituting an index by a logical condition. As the selective masking operations are performed in parallel, the memory can be considered as content addressable, this property is a characteristic of the massively parallel architecture.

FORTRAN PLUS also includes a number of functions to facilitate an efficient use of the DAP. The functions used in the code have the following actions:

matr(V,R)

Returns a matrix by copying the vector V onto R rows (vector broadcasting).

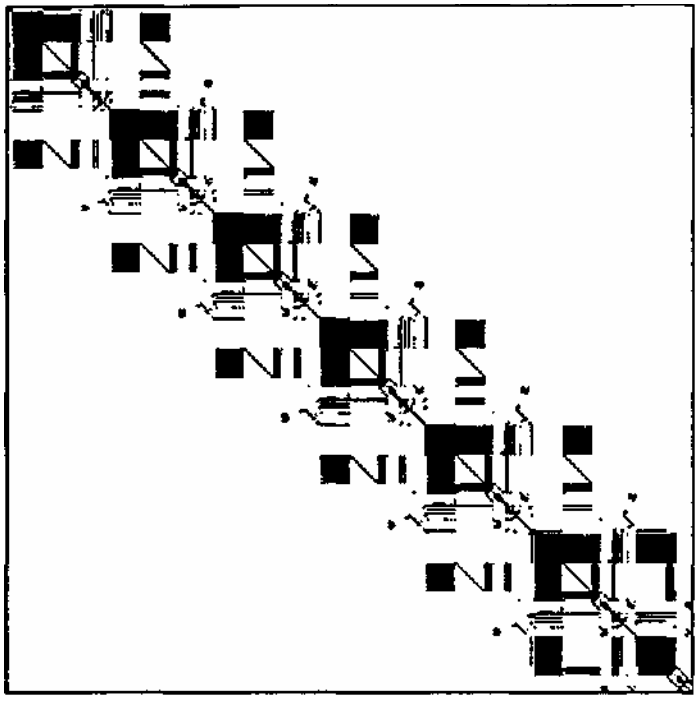
sumc(M)

Returns a vector of the sums over columns of the matrix M.

The following program example shows the time-critical part of the code for the DAP. The pointers to the data structure LTAB1, LTAB2 and KTAB are set up on the host computer.

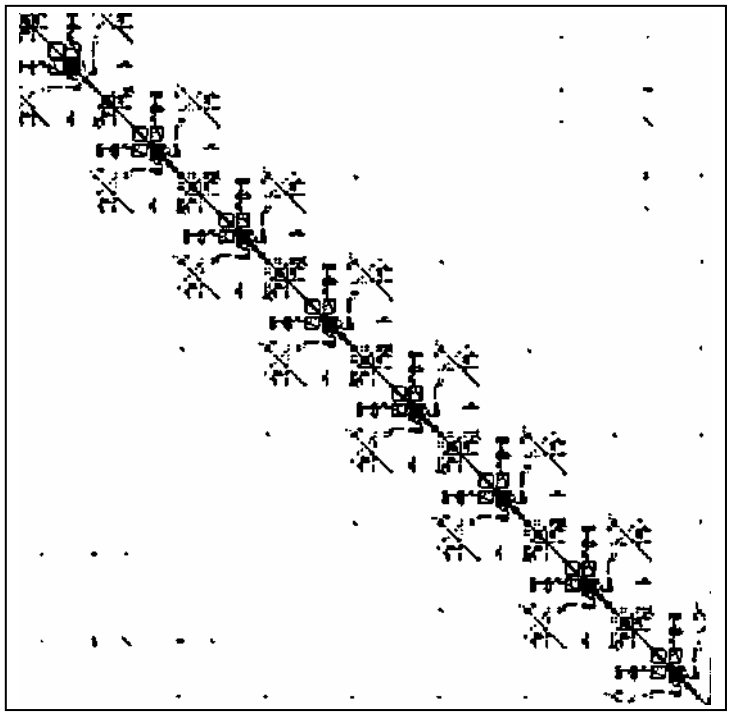
```
            PROGRAMESS
C     ------ MATRIX-CECTOR multiplication scheme.
            Integer P, ROWS, COLUMS
            Parameter(P=32, ROWS=356, COLUMNS=356, NMEM=43,
           >              MBLK=(ROWS-1 )/P+1, NBLK=(COLUMNS-1 )/P+1)
            real A(*P, *P, NMEM), AX(*P, *P, NMEM), BLKSUM(*P, *P)
            real BRCAST(*P,*P), X(*P, NBLK), XBLOCK(*P)
            real XMERG(*P, *P, NMEM), Y(*P, MBLK)
            integer IA, IB, JMAP(*P, *P, NMEM)
            integer K, KTAB(MBLK), L, LTAB1(NBLK), LTAB2(NMEM)
c
c     ------ 1. x-vector broadcasting
c
            do 100 L=1, NBLK
            BRCAST = matr( X(, L), P)
            IA = LTAB1(L)
            IB = LTAB1(L+1)-1
c
            do 100 KA =  IA, IB
            KMEM = LTAB2(KA)
            XMERG(JMAP(, , KMEM).eq.L, KMEM) = BRCAST
          100 continue

c     ------ 2. Parallel multiplication
            do 200 KMEM=1, NMEM
            AX(, , KMEM)= A( , , KMEM)*XMERG(, , KMEM)
          200 continue

c     -----  3. Parallel summation
            do 300 K = 1, MBLK
            IA = KTAB(K)
            IB = KTAB(K+1)-1
            BLKSUM = AX(, , IA)
            do 310 KMEM = IA+1, IB
            BLKSUM = BLKSUM+AX(,, KMEM)
          310 continue
c
            Y( , K)= sumc(BLKSUM)
300 continue
```

Appendix B



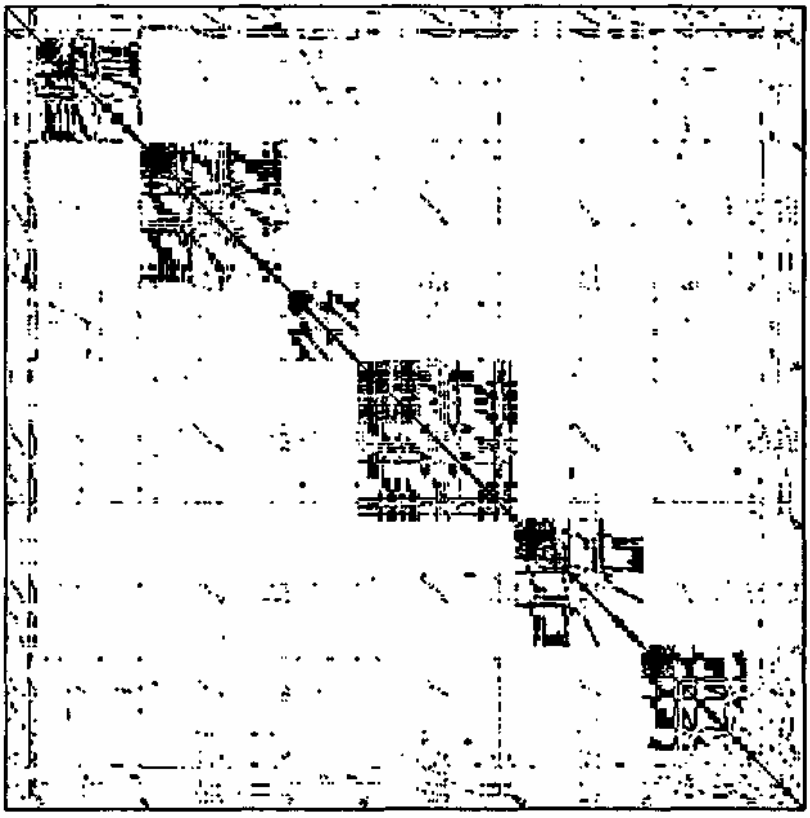
stair 356/1 1275O

Fig. 8.1



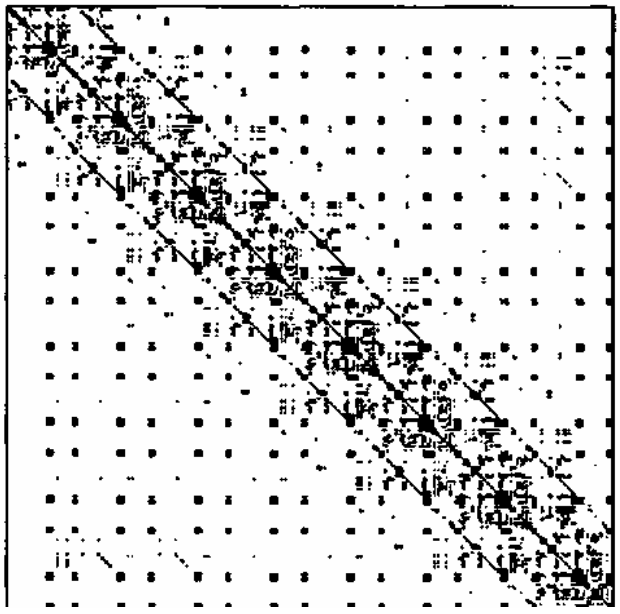
pilotxue 722/2 1O372

Fig. 8.2



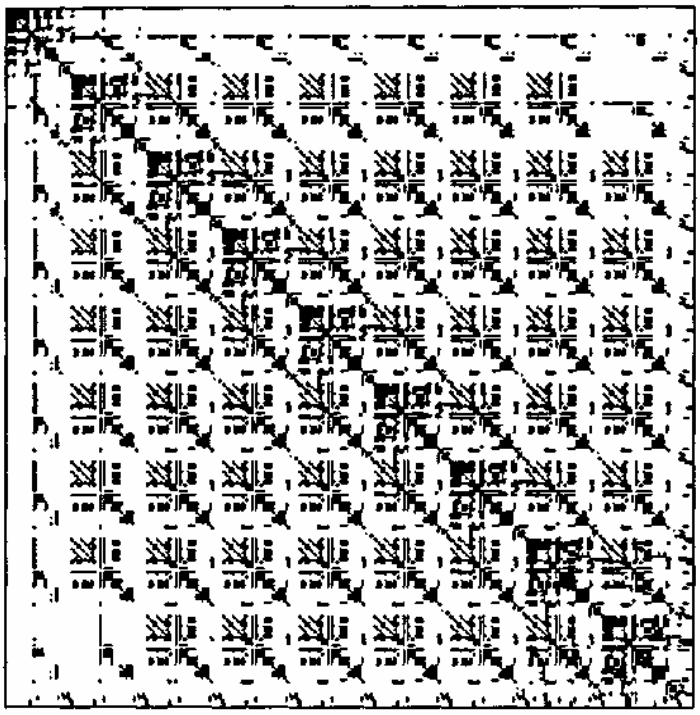
25fv47 82O/2 22968

Fig. 8.3



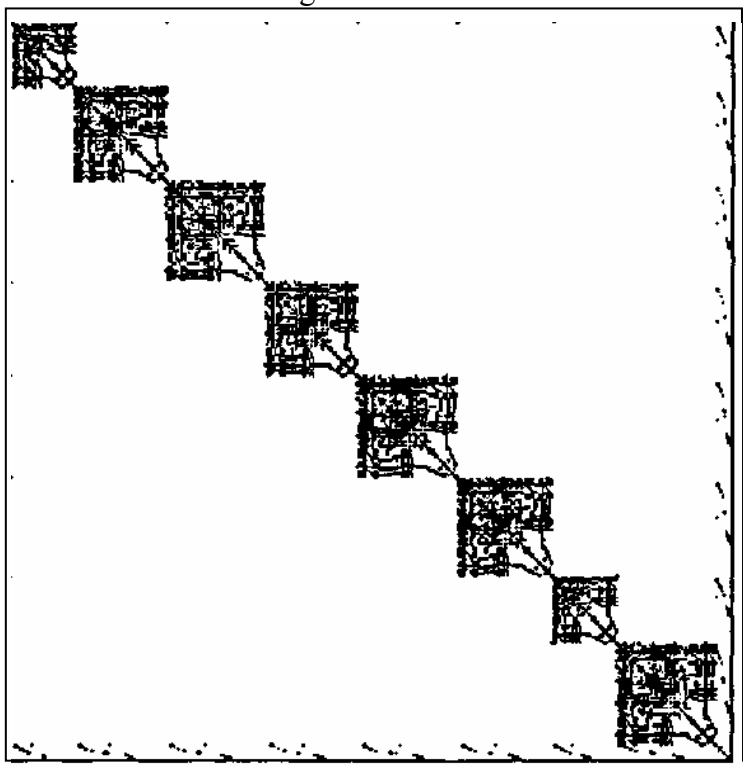
Pilotxja 924/3 27424

Fig. 8.4



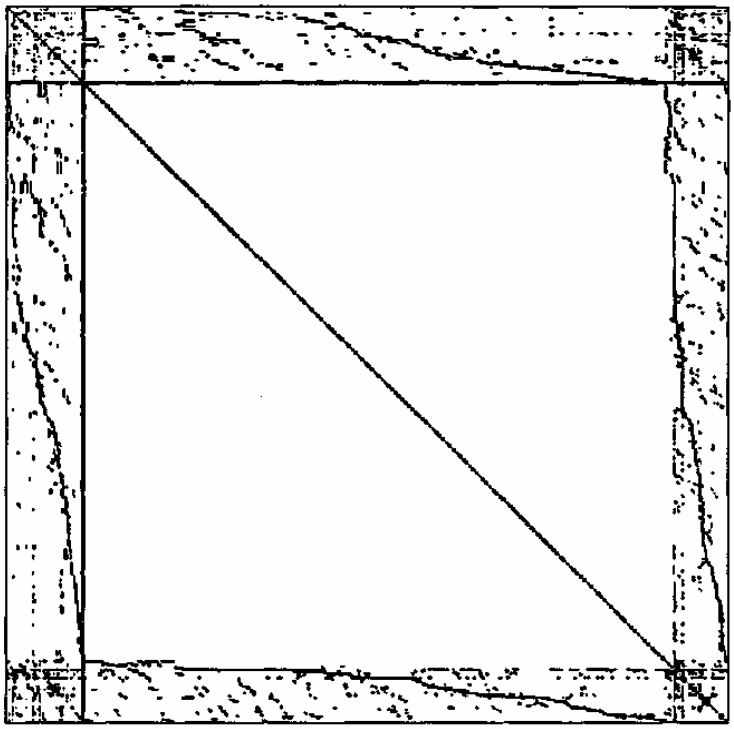
pilot 1441/4 12O521

Fig. 8.5

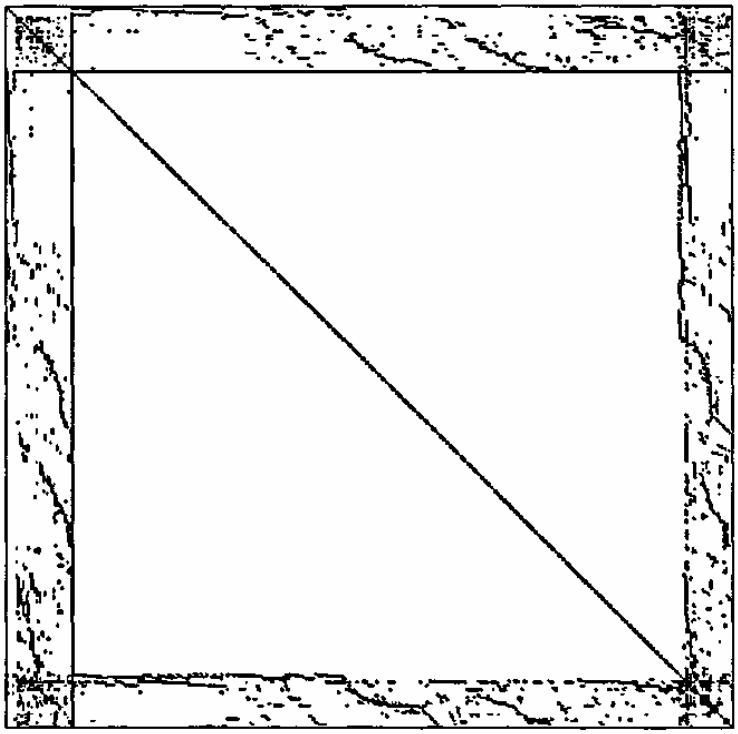


cycle 1889/5 57301

Fig. 8.6



cre_c 2986/7 40796
Fig. 8.7



cre_a 3428/8 44924 Fig. 8.8

9. References

[AMTFOR90] Active Memory Technology (1990), FORTRAN-PLUS enhanced, Technical publication man 102.01. AMT Ltd. Reading, U.K.

[ANLEVM90] Andersen, J., Levkovitz, R., Mitra, G., and Tamiz, M. (1990) Adopting the Interior Point Method for the solution of LP's on serial, coarse grain parallel and massively parallel computers. Technical Report, Dept of Mathematics and Statistics, Brunel University, Uxbridge, Middlesex, U.K.

[BARESH84] Barlow, R.,H., Evans, D.,J., Shanehchi, J., (1984) Sparse Matrix Vector Multiplication on the DAP, Super Computers and Parallel Computation, Clarendon Press, Oxford, pp 149-155.

[CARHKW89] Carolan, W.J., Hill, J.,E., Kennington, J.,L., Niemi, S., Wichman, S..J. (1989) An Empirical Evaluation of the KORBX Algorithms for Military Airlift Applications, Tech. Report 89-OR-06, Dept. Comp. Sci. Southern Methodist University, Dallas, USA.

[ERHEL] ErhelJ., (1990) Sparse Matrix Multiplication on Vector Computers, International Jounal of High Speed Computing, Vol. 2 No.2, June 1990.

[GAYM85] Gay, D.M. (1985) Electronic Mail Distribution of Linear Programming Test Problems, Mathematical Programming Society, COAL Newsletter.

[GEOLIU81] George, J.A. and Liu, J.W., Computer Solutions of Large Sparse Positive Definite Systems, Prentice Hall, 1981.

[KARMAR84] Karmarkar, N. (1984) A New Polynomial Time Algorithm for Linear Programming, Combinatoria, vol 4, pp373-394.

[MORMAK84] Morjaria, M., and Makinson, G. J. (1984) Unstructured Sparse Matrix-Vector Multiplication on the DAP, Super Computers and Parallel Computation, Clarendon Press, Oxford, pp 157-166, 1984.

[PARKINS81] Parkinson, D. (1981) Sparse Matrix Vector Multiplication on the DAP, Technical Report, DAP Support Unit, Queen Mary & Westfield College, London, UK.

[PARKIN90] Parkinson, D., and Litt, J., (editors) (1990) Massively Parallel Computing with the DAP, MIT Press, Cambridge , Massachusetts. U.S.A.