VECTORIZED SPARSE ELIMINATION

D. A. Calahan
Department of Electrical and Computer Engineering
University of Michigan
Ann Arbor, MI 48109

ABSTRACT

Vectorizable sparse equation solution algorithms are classified by the matrix structure which they favor. The state-of-the-art for solution of relatively dense systems is then reviewed. A hybrid vector construct is defined for the increasingly common structure of both moderate local matrix density and global matrix regularity. Estimates are made of CRAY-1 speedup achievable with this construct. A finite difference matrix is studied as an example.

INTRODUCTION

Direct solution of sparse systems has enjoyed wide application to simulation of lumped physical systems described by ordinary differential equations. Also, the last decade has seen a movement toward implicit solution of partial differential equations away from explicit procedures. An excellent example is Navier Stokes aerodynamic simulation codes, which have changed from the purely explicit, through hybrid explicit-implicit and now purely implicit procedures.

The vectorization of direct solution portions of large codes has an immediate aspect related to the recording of specific equation solvers for a particular architecture. Although most vector architectures have at least a minimal provision for sparse vector operations, an overhead is inevitably incurred in reduced memory bandwidth and/or the loading of associated bit maps and linked lists. It is the goal of research in sparse matrix algorithms to reduce this overhead by re-organization of the computation either (1) to obtain longer vectors, or (2) to reduce data flow, and thus achieve an overall speedup.

This paper (1) classifies sparse matrix characteristics amenable to vector processing, (2) reviews the state-of-the-art in solving certain of these problems, and (3) presents new results in the detection of vectors in patterned sparse systems. All of the experimental results were obtained from the CRAY-1; even the algorithm classifications to be made are useful only for a memory-hierarchi- cal processor of the CRAY-1 class with a range of scalar, short vector, and long vector capabilities.

CLASSIFICATION

Consider the linear system $Ax = b$ solved by triangular factorization of $A$ into $L$ and $U$. Assume that the factorization has proceeded by outer product column-row operations so that an $mn$ unreduced system remains. The structure of this unreduced system— which includes fill from the completed portion of the reduction—then becomes the principal issue in determination of the sparsity algorithm to be used during the remainder of the reduction. This is an important generalization beyond examination of only the structure of $A$, since it suggests the use of different algorithms (polyalgorithms) as the reduction proceeds and fill increases the density of the unreduced portion.

Four sparsity structures will be considered at various parts of this paper; they are listed below to assist in unifying the later discussion. These distinguishing
Vectorizable sparse equation solution algorithms are classified by the matrix structure which they favor. The state-of-the-art for solution of relatively dense systems is then reviewed. A hybrid vector construct is defined for the increasingly common structure of both moderate local matrix density and global matrix regularity. Estimates are made of CRAY-1 speedup achievable with this construct. A finite difference matrix is studied as an example.
attributes are related to local and global sparsity characteristics:
(a) locally and globally dense, partitioned;
(b) locally dense, globally unpatterned;
(c) locally dense, globally patterned;
(d) locally sparse, globally patterned;
(e) locally sparse, globally unpatterned.
The last is the least vectorizable. Its scalar solution is probably amenable to speedup only by using a MIMD architecture and so will not be discussed further.

**BLOCK-ORIENTED SPARSE SOLUTION**

**INTRODUCTION**

Two classes of relatively dense matrices benefit from solution by a general sparse solver which is oriented toward the solution of block structures. Although algorithmically less challenging than the sparser case to be studied later, such structures are becoming more common due to the aforementioned increase in the implicitness of PDE solution codes.

**THE DENSE, PARTITIONED CASE**

The utility of a general sparse solver in the analysis of full, banded, and other dense systems arises from vector length limitations of the processor, which in turn results from a relatively small cache memory in a hierarchical memory system. Such dense systems must be block-partitioned; in the case of the CRAY-1, these partitions must be limited in one dimension to 64, the maximum vector length of the machine. Using a general solver avoids the writing of specialized assembly language routines for dense systems with globally different density patterns but which are partitioned into locally similar 64-length or smaller dense blocks.

The processing of such large blocks with a sparse solver can be carried out on the CRAY-1 with >99% of the solution time in numeric kernels, and with <1% in processing of lists resulting from the general sparsity assumption. A variety of common compressed storage schemes can also be accommodated.

**LOCAL DENSITY**

Moderate-sized dense blocks occur naturally from the representation of variable and equation coupling, from nodal coupling in a grid, and from coordinate transformations, among other causes. In the absence of other vectorization strategies (to be discussed shortly), it becomes necessary to reduce the system a block at a time with dense matrix kernels of a block-oriented sparse solver. Descriptors of the location and size of the block suffice to guide the solution of such a system.

The overhead of list processing of the blocks may be compensated by finely-tuned numeric kernels, with the net result that a general solver can execute at a higher rate than a conventionally-coded specialized solver.

The execution rate is of course highly dependent on the matrix sparsity structure. However, a timing model of the numeric kernels and the list processing overhead allows the establishment of MFLOPS bounds for matrices of constant block sizes but arbitrary block sparsity patterns. Such bounds are given in Table 1 for the CRAY-1. The minimum rate is achieved with a single off-diagonal block (e.g., block tridiagonal) and the maximum with r off-diagonal blocks (Figure 1), as r → ∞.

**LOCALLY DENSE, GLOBALLY PATTERNED SPARSE SYSTEMS**

**INTRODUCTION**

Table 1 shows that processing block sizes with dimensions below 10 utilizes a small fraction of the CRAY-1 processor speed. To regain a high processing rate, another structural property be-
to use the notion of the graph of a matrix.

The non-zero structure of a matrix \( A \), where \( A \) is structurally symmetric, has a convenient graph theoretic formulation. Assume that \( a_{ij} \neq 0 \), \( i=1,2,...,n \). Let \( V = \{v_1,v_2,...,v_n\} \) with the \( v_i \) termed vertices and \( V \) the vertex set. Define a set \( P \) of ordered pairs of \( V \), called edges, by \((v_i,v_j) \in P \) if and only if \( a_{ij} \neq 0 \) and \( k \neq j \). Then \( G=(V,P) \) is called the graph of \( A \).

Table 1. Performance of general block sparse system solver on the CRAY-1

<table>
<thead>
<tr>
<th>Block sizes</th>
<th>MFLOPS range</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.9 - 7.6</td>
</tr>
<tr>
<td>3</td>
<td>5.0 - 17.0</td>
</tr>
<tr>
<td>4</td>
<td>10.0 - 26.0</td>
</tr>
<tr>
<td>6</td>
<td>21.0 - 43.0</td>
</tr>
<tr>
<td>8</td>
<td>32.0 - 60.0</td>
</tr>
<tr>
<td>12</td>
<td>54.0 - 84.0</td>
</tr>
<tr>
<td>16</td>
<td>69.0 - 98.0</td>
</tr>
<tr>
<td>32</td>
<td>102.0 - 124.0</td>
</tr>
<tr>
<td>64</td>
<td>126.0 - 141.0</td>
</tr>
</tbody>
</table>

To illustrate the relationship between local and global properties, consider the subgraphs \( G_1 \) and \( G_2 \) of Figure 2(a). These subgraphs are possibly connected by paths through vertices not shown, but are assumed to be not directly connected. If the associated equations are arranged in the numbered order, the partial matrix structure of Figure 2(b) results. This structure is locally dense (contrast full) but globally sparse, since the two dense sub-matrices are not coupled in the northwest matrix partition. If the equations are reordered so that similarly-connected nodes are consecutively ordered, then each of the resulting 16 partitions is either a diagonal or null sub-matrix (Figure 2(c)). Because most sparse blocks are coupled to other sparse blocks by diagonal coupling blocks, the matrix structure is now termed globally dense. (It may be noted that the local density pattern of each dense block of Figure 2(b) is identical to the global density pattern of Figure 2(c).) The factorization of the northwest corner of the system matrix may utilize any algorithm, independently of the algorithms used to reduce the remainder of the matrix.

If the connection symmetry between the two sets of nodes undergoing reduction extends to their interconnections to other unreduced nodes as in Figure 3(a), and if these unreduced nodes are properly ordered as shown, then the northeast and southwest parti-
Figure 2. Relationships between local, global matrix properties

Figure 3. Similar subgraphs with similar connections to rest of graph.
torization of a diagonal block, (b) r block forward and back substitutions, and (c) \(r^2\) multiplications/accumulations. For \(r=3\) (a common number for dissected finite element and finite difference grids), 69\% of the operations are of the M/A type. The M/A kernel therefore warrants principle study.

The nature of the M/A model kernel with both local diagonal sparsity and global density is illustrated in Figure 4. It is proposed to study the execution of the kernel

\[
C \rightarrow C \rightarrow A \times B
\]  

(1)

where \(A\), \(B\), and \(C\) are illustrated in the figure.

\[
\begin{bmatrix}
  A_1x_1 \\
  A_2x_2 \\
  A_3x_3
\end{bmatrix} \times \begin{bmatrix}
  B_1x_1 \\
  B_2x_2 \\
  B_3x_3
\end{bmatrix} = \begin{bmatrix}
  C_1x_1 \\
  C_2x_2 \\
  C_3x_3
\end{bmatrix}
\]  

(2)

represents, on equation reordering, the simultaneous multiplication of four \(3 \times 3\) full matrices. It is proposed to implement the associated accumulation kernel by forming

\[
\begin{bmatrix}
  C_{1j} \\
  C_{2j} \\
  C_{3j}
\end{bmatrix} = \sum_{k=1}^{3} \begin{bmatrix}
  A_{1k} \\
  A_{2k} \\
  A_{3k}
\end{bmatrix} B_{kj}
\]  

(3)

i.e., by accumulating a column of diagonal blocks of \(C\). To perform each term of the summation by a single chained multiply-add vector operation with the CRAY-I requires chain replications of the \(4\)-length \(B_{kj}\) to a \(12\)-length vector so that the overhead of the replication does not seriously impact the overall timing.

The basis of this replication is the recursive feature of the vector logical pipeline, whereby, if the same vector register is both operand and result register—usually prohibited in register allocation—data will be delayed four clocks in the pipeline and the desired replication achieved. Figure 5 gives the CAL instruction sequence and the clock level report of a part of the accumulation loop, as reproduced by a CRAY-I timing simulator.

The preference for processing hybrid kernels can be expected to arise from the interconnection or, more generally, the data flow protocol of the processor. For the CRAY-I, two recursive features of the vector registers permit high performance M/A kernels.

4-Matrix M/A. The pattern illustrated by the matrix multiply
but suffers from register and pipeline overhead resulting from four separate invocations of the full matrix multiply. Table 2 shows that nearly three times the execution rate is achieved for multiplication of four 4x4 matrices with the specialized kernel, in comparison with the standard CAL kernel.

8 Matrix M/A. A similar recursive feature of the addition pipeline allows the rapid accumulation of 8-length vectors and consequently the simultaneous multiplication of 8 matrices. This is a well known feature described in [8] and will not be discussed here. It suffices to note in Table 2 the extraordinary speedups achievable with very small matrices. However, the execution rate has a large discontinuity between n=7 and n=8, due to the nature of the algorithm, and is less desirable beyond n=7 than a 4-matrix multiply.

<table>
<thead>
<tr>
<th>T</th>
<th>A INSTRUCTION</th>
<th>8-ADDR</th>
<th>CP</th>
<th>PPPVVV</th>
<th>FFF</th>
<th>V. REG</th>
<th>BSRRR</th>
<th>SS</th>
<th>REG</th>
<th>A A. REG</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>V1, A0+A0</td>
<td>54C</td>
<td>629:00Z</td>
<td>50 00 15</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>A3 A0+A3</td>
<td>54D</td>
<td>629:00Z</td>
<td>50 00 15</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>A3 A3+A1</td>
<td>55A</td>
<td>631:00Z</td>
<td>50 00 15</td>
<td>4</td>
<td>14</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>V1 A2</td>
<td>55B</td>
<td>632:00Z</td>
<td>50 00 15</td>
<td>4</td>
<td>17</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>A0 A0+A5</td>
<td>55C</td>
<td>633:00Z</td>
<td>50 00 15</td>
<td>4</td>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>A5 A5+A6</td>
<td>55D</td>
<td>634:00Z</td>
<td>50 00 15</td>
<td>4</td>
<td></td>
<td>199</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>A5 A0+A5</td>
<td>56A</td>
<td>636:00Z</td>
<td>50 00 15</td>
<td>4</td>
<td></td>
<td>1A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>V0 A0+A0</td>
<td>56B</td>
<td>637:00Z</td>
<td>50 00 15</td>
<td>4</td>
<td></td>
<td>1B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>V3 V3 V3 VM</td>
<td>56C</td>
<td>642:00Z</td>
<td>50 00 15</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>V4 V3 V0</td>
<td>56D</td>
<td>644:00Z</td>
<td>50 00 15</td>
<td>4</td>
<td>1F</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>A0 A0+A4</td>
<td>57A</td>
<td>646:00Z</td>
<td>50 00 15</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>A4 A4 A4</td>
<td>57B</td>
<td>648:00Z</td>
<td>50 00 15</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>V6 V7 V4</td>
<td>57C</td>
<td>650:00Z</td>
<td>50 00 15</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>V1 A1</td>
<td>57D</td>
<td>651:00Z</td>
<td>50 00 15</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>J V1 A0+A0</td>
<td>60A</td>
<td>652:00Z</td>
<td>50 00 15</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5. Simulator output for 4-matrix accumulation instruction sequence for VL = 64. V1 is replicated into V3 with VM = 0077...8.
A number of algorithmic questions will be left unanswered, a topic for continuing research.

**GENERATION OF HYBRID VECTORS**

**INTRODUCTION**

Given the graph of a matrix, it is proposed to perform operations on this graph which yield hybrid vectors in the matrix reduction with either no increase or a determinable increase in the arithmetic operation count. The example of a 5-point 2-D finite difference grid will be used to illustrate the procedure, because of its connection regularity and because its solution by nested dissection is characterized by exploitation of decoupling to achieve a reduced arithmetic operation count for large grids. The reader is assumed to be familiar with this dissection process.

**FOLDING AND ROTATION**

The (diagonal) nested dissection of a 5x5 grid proceeds by recursively dividing the grid into quadrants until each quadrant consists of a single node. This division is performed along diagonal separators, which are lines of nodes whose removal divides the graph into unconnected parts.

It is clear from Figure 6(a) that, since the quadrants have a similar structure, "similarly-positioned" vertices not on a separator may be eliminated simultaneously with vector operations, without increasing arithmetic computation. These vectors will be of length four, as required for the 4-matrix kernel of Figure 5.

"Similarly-positioned" nodes can be generated by overlaying the quadrants so that a single node in the overlay represents 4 nodes. This single-quadrant representation of the 4 quadrants may be achieved by folding or rotating the original graph. This rotation process is illustrated in Figure 6(a)-(b); a recursive folding process which generates vectors of decreasing length is discussed in ref. [14].
Because the non-separator nodes are eliminated first in the nested dissection process, these interior nodes are represented by the northwest corner of the system matrix; this corner is consequently guaranteed to consist of $4 \times 4$ blocks with either diagonal or null structure. The southeast corner of $L$, representing the reduction of the separator nodes, is dense and can be reduced at execution rates exceeding 100 MFLOPS. The northeast and southwest partitions, however, represent coupling between the separators and the interior nodes of the quadrants. Asymptotically in the grid dimensions, operations involving these two partitions consist of approximately $30\%$ of the total, so that the choice of a proper kernel is important (Figure 7). Irregularity in these coupling matrices results in part from the separator nodes shared by $Q_1$ and $Q_4$ (nodes #1, #7 and #13 in Figure 6(a)) in the rotation sequence. The regularity may be restored by cutting the graph along the boundary, adding nodes and associated unknowns, and adding equations that relate the new nodes to the originally shared ones. This cutting process is illustrated in Figure 6(c)-(d).

The structure of $L$ and $U$ resulting from application of such cutting to a $17 \times 17$ finite difference grid is shown in Figure 8(b). The conventional nested dissection ordering for the same matrix yields the $LU$ map of Figure 8(a). Coupling in the northwest partition appears as $4 \times 4$ diagonal blocks, as predicted; coupling in the northeast and southwest partitions appears as $4$-length stripes, but not necessarily as $4 \times 4$ diagonal blocks. Thus a somewhat modified $4$-matrix accumulation kernel would have to be used. The addition of the 8 nodes along the cut also increases each dimension of the dense southeast corner of $LU$ by approximately $25\%$. The total increase in computation resulting from cutting is as yet undetermined.

**SUMMARY**

Vectorization and data flow 

![Figure 6. Rotation of quadrants (a) into single-quadrant representation (b); rotation with cut and creation of nodes ((b)-(c)).](image)
for a memory hierarchical process— or add two new issues to be con-
idered in the development of codes
for the direct solution of 2-D
finite difference grids. What is
a single algorithm class— nested
dissection—for a scalar machine
now divides into subclasses of
algorithms, of which the above
proposal is only one. Checker-
board and related ordering strat-
egies are also attractive; pre-
liminary estimates indicate such
codes will execute over 100 MFLOPS
on the CRAY-1S, which can partial-
ly compensate for increased arith-
metic computation.

\[
\begin{array}{|c|c|c|}
\hline
\text{accum: 4-matrix} & \% \text{opns: 50} & \text{MFLOPS: Table 2} \\
\hline
\text{accum: dense} & \% \text{opns: 20} & \text{MFLOPS: >100} \\
\hline
\end{array}
\]

*Approx. percent of total arith-
metic operations.

Figure 7. Estimated asymptotic
perf. of polyalgorithm
to perform nested dis-
section in four matrix
partitions.

SPARSE, PATTERNED SYSTEMS

As the coupling in A, B, and
C of Figure 4 decreases, each
approaches a diagonal matrix. The
accumulation then involves at least
two vector loads (and usually one
vector store) for each floating
point M/A operation and sufficient
list processing to locate at least
two of the matrices in memory. An
accumulation kernel written for
the CRAY-1, including list process-
ing and a vector store for each
accumulation, executes at the rate

\[
\text{MFLOPS} = 53.3 \left(1 - \frac{1}{1 + 31.3/\ell}\right)
\]

with the maximum value of 35.8 for
\( \ell = 64 \). This is less than 1/4 the
asymptotic rate of a dense accumu-
lation. The kernel is memory bound
and involves significant start up
time for the relatively small float-
ing point computation involved.

CONCLUSION

While the speed of vector
processors encourages the formul-
ation of denser systems, their in-
creasingly parallel design favors
the construction of longer vectors
that can be distributed across
many pipelines operating concur-
rently. In this paper, the vector-
lengthening advantages of the hy-
brid vector construct have been
shown at the kernel level and
methods have been proposed to pro-
duce such vectors directly from
the problem structure.

From the algorithm viewpoint,
the direct relationship between
problem and processor structure
offers novel insight possibly use-
ful in developing a family of
equation ordering techniques
based on folding, rotation, etc.
It is also hoped that a high per-
formance software package may be
developed for specific 2D grid
geometries.

From the viewpoint of process-
or architecture, this paper has
quantified the motion that the
less dense the system, the more
data flow and other accumulation
kernel overhead is required. A
patterned system may permit the
lengthening of vectors—which re-
duces the influence of overhead—but does not significantly alter
the data flow problem.

ACKNOWLEDGMENT

The author acknowledges the
programming assistance of M.
Yatchman, and the insight provided
by a CRAY-1 simulator developed by
D. A. Orbits. This work was spon-
sored jointly by the Mathematical
and Information Sciences Director-
ate of the Air Force Office of
Scientific Research, and by the
Air Force Flight Dynamics Labora-
tory, Wright Patterson AFB under
Figure 8(a). Matrix of dissected 17x17 5-point finite difference grid; before rotation.
Figure 8(b). Matrix of dissected 17x17 5-point finite difference grid; after rotation.
Grant 75-2812.

References


