

## THE IMPACT OF HARDWARE GATHER/SCATTER ON SPARSE GAUSSIAN ELIMINATION\*

JOHN G. LEWIS† AND HORST D. SIMON†

**Abstract.** Recent vector supercomputers provide vector memory access to "randomly" indexed vectors, whereas early vector supercomputers required contiguously or regularly indexed vectors. This additional capability, known as "hardware gather/scatter," can be used to great effect in general sparse Gaussian elimination. In this note we present some examples that show the impact of this change in hardware on the choice of algorithms for sparse Gaussian elimination. Common folk wisdom holds that general sparse Gaussian elimination algorithms do not perform well on vector computers. Our numerical results demonstrate that hardware gather/scatter allows general sparse elimination algorithms to outperform algorithms based on a band, envelope, or block structure on such computers.

**Key words.** sparse Gaussian elimination, sparse matrices, reordering algorithms, vector computers, vectorization, hardware gather/scatter

**AMS(MOS) subject classification.** 65F05

**Background.** Early experience with sparse Gaussian elimination on vector computers [3] showed none of the dramatic improvements in speedup encountered in other linear algebra computations. This is due to the fact that Gaussian elimination with a sparse data structure requires access to irregularly spaced data. Early vector computers, such as the CRAY-1 and the CYBER 205 computers, allow vector memory transfers only for contiguously or regularly spaced vectors. Most sparse Gaussian elimination algorithms spend the vast majority of the factorization execution time in a loop of the following type:

```
INTEGER I, N, M
INTEGER INDEX (M)
REAL A, X(M), Y(N)
  :
DO 10 I=1, M
  Y(INDEX (I))= A * X(I)+ Y(INDEX I)
10 CONTINUE
```

The indexing or indirect addressing for the vector  $Y$  creates irregular spacing in the data and prevents use of the vector arithmetic units on early vector computers.

This loop is often referred to as a sparse or indexed SAXPY. The efficiency of the implementation of this loop determines the performance of the sparse factorization algorithm. Because of the importance of this loop, or kernel, a subroutine called SAXPYI, which follows the spirit and the notation of the BLAS [6], has been proposed as a facility in extensions of the BLAS [2].

The use of the SAXPYI loop as presented above, in FORTRAN, would result in no use of the vector hardware of early supercomputers. The loop would be executed using scalar instructions only, producing no speedup at all over equivalent scalar computers. Indeed, the indexing of the result vector  $Y$  prevents vectorization on any existing vector computer unless it is known that the indices are distinct. Even on current

\* Received by the editors June 23, 1986; accepted for publication (in revised form) April 15, 1987.

† Boeing Computer Services, Engineering Technology Applications, Seattle, Washington 98124.

vector computers, the SAXPYI loop does not vectorize without using directives to the compiler.

When the indices are known to be distinct the indexed SAXPY loop can be rewritten to allow some use of the vector hardware on any vector computer. Dembart and Neves [1] analyzed seven different formulations of this loop on a CDC STAR 100, and determined that there were combinations of vector length and vector density for which each of the formulations was fastest. Similar analyses by these authors for the CYBER 203 and 205 showed corresponding results, although the ratios changed. For reasonable combinations of vector length and vector density the most important of the six alternatives to the original scalar code was the same on all three machines. This alternative is:

```

C
C      gather the sparse elements into a dense vector
C
      DO 100 I = 1, M
          TEMP (I) = Y(INDEX (I))
100 CONTINUE

C
C      perform a vectorizable dense SAXPY
C
      DO 200 I = 1, M
          TEMP (I) = TEMP (I) + A * X(I)
200 CONTINUE

C
C      scatter the elements back into the sparse vector
C
      DO 300 I = 1, M
          Y(INDEX (I)) = TEMP (I)
300 CONTINUE

```

Although this looks far more complicated than the original loop, it permits the use of the vector arithmetic units for the numerical loop. The nonvectorizable memory transfers are isolated to separate loops that could be made more efficient by using assembly language subroutines (albeit in scalar mode). This formulation of the indexed SAXPY is known as a GATHER-SAXPY-SCATTER (GSS) implementation, for the operations performed in turn by the three loops.

On a CRAY-1 computer, the original indexed SAXPY loop written in FORTRAN executes at a maximum rate of about 4 megaflops, much less than the maximum rate of 155 megaflops this machine can achieve for other operations. Woo and Levesque [11] analyzed the GSS formulation and showed that its maximum rate in assembly language was around 8 megaflops. Alternatively, a good assembly language implementation of the original loop that uses only the scalar hardware performs asymptotically at 13 megaflops (see [10]). This implementation is never slower than the GSS formulation, demonstrating that the SAXPYI operation is essentially a scalar computation for the CRAY-1.

In contrast, the standard implementations of banded or variable banded Gaussian elimination algorithms use dense SAXPY operations or dense dot products as their fundamental inner-loops. Such implementations can achieve vector speeds on vector computers. However, they usually do not approach the asymptotic speeds of these

machines because the vector lengths are limited by the bandwidth, which should not become very large. Still, the possibility of using the vector hardware for these schemes *and the inherent performance limitation of the indexed SAXPY loop* has led to the conventional folk wisdom that (variable) banded factorization schemes will usually outperform general sparse Gaussian elimination on vector computers.

The vector supercomputers being produced currently, in particular the CRAY X-MP/4 and the more recent models of the CRAY X-MP/2, are equipped with hardware facilities that permit memory access according to an index vector. That is, these machines permit the GATHER and SCATTER loops to be performed using *vector* memory transfers to and from a hardware vector register. This gather/scatter hardware leads to a much faster implementation of SAXPYI by using the GSS formulation. The assembly language coded implementation in VectorPak reaches 78 megaflops asymptotically. Some detailed SAXPYI timings are given in Table 1 below. These results are reported in [8] and [10], and were obtained by averaging over 50 executions of each loop. In each case the index vector INDEX was set as follows: INDEX (1) =  $N$ , INDEX (2) =  $N - 1$ ,  $\dots$  INDEX ( $M$ ) =  $N - M + 1$ .

The older CRAY Fortran Compiler CFT 1.13 does not make use of the hardware for gather/scatter. A corresponding VectorPak implementation of SAXPYI has been developed for CRAY X-MP's without hardware gather/scatter. Both exhibit the scalar performance characteristic of the CRAY-1. In contrast the utilization of hardware gather/scatter either with a compiler directive to CFT 1.14 or with VectorPak shows a dramatic improvement.

**Numerical results.** We present two sets of numerical results. The first isolates the effects of the gather/scatter hardware by comparing the performance of a CRAY X-MP/24 with this hardware to the performance of an X-MP/24 without this hardware. The second set of results compares the performance of an X-MP/24 with hardware gather/scatter to a CRAY 1-S (which had no option for such hardware). Together these results demonstrate that this limited change in the hardware of vector computers changes the algorithm of choice for many sparse problems.

Our first series of numerical results demonstrates the speed-up which can be obtained in very large problems taken from several engineering applications by using hardware for gather and scatter in a general sparse elimination algorithm. We used the modified minimum degree (MD) algorithm by Liu [7] to reorder seven large problems taken from the Harwell/Boeing sparse matrix collection [4]. A short description of each problem is given in Table 2 and corresponding ordering statistics are given in Table 3. All problems, with the exception of LRGPWR, are finite element models of large three-dimensional structures. All matrices are symmetric and positive definite.

TABLE 1  
SAXPYI Speed on the CRAY X-MP/24 using 1 CPU.  
(Rates given in megaflops.)

	$M = 10$	$M = \text{infinity}$
(ignoring the hardware for gather/scatter)		
CFT 1.13	5.0	5.7
VectorPak	6.3	14.5
(using the hardware for gather/scatter)		
CFT 1.14	16.1	54.6
VectorPak	16.1	78.6

TABLE 2  
Problem description.

Problem	Description
STK3562	Calgary Winter Olympics Coliseum (Olympic Saddledome)
STK3948	Offshore oil platform
STK 4884	Corps of Engineers model of dam
LRGPWR	Electric power network of U.S.
ST10974	Elevated pressure vessel
ST11948	R. E. Ginna nuclear power station
ST15439	Columbia Center (76 story skyscraper)

TABLE 3  
Fill-in characteristics of matrices in Table 2.  
(Multiple minimum degree ordering.)

Problem	$N$	$A_{nz}$	$L_{nz}$	$L_{subs}$	$F_{ops}$	$S_{ops}$
STK3562	3,562	78,174	275,360	22,036	16,355,839	554,282
STK3948	3,948	56,934	647,274	59,935	82,845,152	1,298,496
STK4884	4,884	142,747	736,294	49,812	74,923,505	1,477,472
LRGPWR	5,300	8,271	22,764	16,796	142,921	50,828
ST10974	10,974	208,838	994,885	92,969	72,642,932	2,000,744
ST11948	11,948	68,571	650,777	102,569	70,911,248	1,313,502
ST15439	15,439	118,401	1401,129	179,537	143,983,290	2,833,136

The columns are as follows:

$N$	Order of the matrix,
$A_{nz}$	Nonzeros in lower triangle of the original matrix,
$L_{nz}$	Nonzeros in the Cholesky factor of $A$ ,
$L_{subs}$	Number of distinct subscripts required for compressed subscripts,
$F_{ops}$	Number of multiply-add pairs performed during factorization,
$S_{ops}$	Number of multiply-add pairs performed during solve.

The numbers in Table 3 show that the factors of all but the power network problem LRGPWR have a substantial number of nonzeros per row. We expect to realize much of the theoretical speedups due to hardware gather/scatter on these practical applications.

All problems were solved using four different implementations of the key SAXPYI loop. Two implementations were derived from a single FORTRAN code created by modifying the original source code by the insertion of a few compiler directives to effect the vectorization of the key loops. (As discussed earlier, the compiler cannot assume the indices are distinct unless told so.) The FORTRAN code with inserted compiler directives was then compiled twice under CFT 1.14. Using a compiler option, code was generated separately for a CRAY X-MP *with* hardware gather/scatter and for a CRAY X-MP *without* hardware gather/scatter.

Two further implementations were derived by replacing the FORTRAN SAXPYI loops with calls to an optimized CRAY assembly language implementation of SAXPYI from VectorPak [10]. This modified code was also compiled under CFT 1.14 and executed twice, using the VectorPak library for CRAY X-MP's without gather/scatter and then the corresponding library for machines with gather/scatter. Tables 4 and 5 present the execution times obtained for factorization and solution for the four implementations. All execution times are listed relative to the time obtained calling VectorPak for machines with gather/scatter. The actual execution time in seconds for

TABLE 4  
*Relative execution times for sparse matrix factorization.*  
 (Normalized so that VectorPak with  $g/s = 1.00$ .)

Problem	CFT 1.14 no $g/s$	CFT 1.14 with $g/s$	VectorPak no $g/s$
STK3562	6.33	1.14	2.54
STK3948	9.66	1.24	3.25
STK4884	8.70	1.20	3.14
LRGPWR	1.25	0.93	1.17
ST10974	7.24	1.16	2.80
ST11948	8.75	1.22	3.12
ST15439	8.78	1.25	3.15

TABLE 5  
*Relative execution times for sparse forward and back substitution.*  
 (Normalized so that VectorPak with  $g/s = 1.00$ .)

Problem	CFT 1.14 no $g/s$	CFT 1.14 with $g/s$	VectorPak no $g/s$
STK3562	6.94	1.48	2.55
STK3948	9.22	1.47	3.13
STK4884	8.77	1.39	3.00
LRGPWR	1.96	1.57	1.11
ST10974	7.45	1.48	2.68
ST11948	5.94	1.50	2.23
ST15439	7.54	1.49	2.68

the implementation that calls VectorPak for hardware gather/scatter is given in Table 6, together with the execution times for an envelope factorization based on the reverse Cuthill-McKee (RCM) algorithm from [5]. This envelope factorization has been optimized by calling the assembly language implementation of SDOT in VectorPak.

Tables 4 and 5 show clearly the direct benefits of the hardware gather/scatter feature for general sparse elimination schemes. The factorization and solution of the structures problems are in some cases almost an order of magnitude faster. Only the very sparse network problem benefits little because the number of nonzeros per row is too small. (The factored matrix from LRG PWR has only about 9 nonzeros per row on average.)

The comparison in Table 6 shows that, contrary to standard expectations, general sparse methods can outperform envelope solvers on vector computers. Because of the natural vectorization of envelope methods and the essentially scalar performance of general sparse methods on earlier vector computers, general sparse methods were commonly thought to be noncompetitive on vector computers. Table 6 disproves this assertion.

Our second set of numerical tests was generated to evaluate the impact of the changing architecture on the choice of numerical algorithms for sparse Gaussian elimination by comparing performance on a CRAY-1S (always without hardware gather/scatter) to a CRAY X-MP with hardware gather/scatter. We solved linear systems with five test matrices arising in reservoir simulation. These matrices were proposed as benchmark problems by Sherman [9] and are available through the sparse matrix collection [4]. All matrices are block seven diagonal unsymmetric matrices

TABLE 6  
*Execution times (sec) for factorization and solution routines.*

Problem	Factorization time		Solution time	
	RCM	MD	RCM	MD
STK3562	3.421	1.276	0.047	0.033
STK3948	7.487	4.074	0.064	0.055
STK4884	4.071	4.129	0.065	0.066
LRGPWR	3.642	0.102	0.061	0.028
ST10974	*	4.891	*	0.108
ST11948	*	3.871	*	0.094
ST15439	17.440	7.791	0.213	0.152

\* Required more than 4,000,000 words of memory.

arising from reservoir models on three-dimensional grids. We solved the five linear systems with each of the five different solution algorithms in SPARSPAK [5]. The reverse Cuthill-McKee algorithm (RCM) is an envelope scheme, which vectorizes well using inner products. Both the one-way dissection (OWD) and refined quotient tree (RQT) algorithms use a block partitioning of the coefficient matrix. Much of the computation in the block elimination scheme also vectorizes with inner products, though the vector length is generally shorter than in RCM. Finally automated nested dissection (AND) and quotient minimum degree (QMD) algorithms use a general sparse data structure and SAXPYI as the inner loop for the factorization. In order to improve performance, computationally intensive sections of the code in SPARSPAK were replaced by calls to VectorPak subroutines, in particular by calls to SDOT and SAXPYI.

We obtained execution times from both the CRAY-1S and on the CRAY X-MP/24 with hardware gather/scatter. In the tables below we denote the order of the problem by  $N$  and the total number of nonzeros in the upper triangular part of the matrix by  $NZ$ .

The average speedup for these five examples is as follows:

RCM	1.05
OWD	1.14
RQT	1.17
AND	2.70
QMD	2.74

Obviously hardware gather/scatter has a dramatic impact on the performance of the general sparse codes (AND and QMD). Both algorithms perform three times faster than on the CRAY-1S and now are the fastest algorithms among the ones considered here. The relative performance of the five algorithms on the CRAY X-MP is about the same as on a scalar machine: the general sparse algorithms, which attempt to minimize the overall fill-in in Gaussian elimination and thus generally perform the least amount of arithmetic, are the most efficient.

The block partitioning methods, OWD and RQT, execute faster on the CRAY X-MP, but their relative speedup is simply due to the faster clock rate on the CRAY X-MP and unrelated to hardware gather/scatter. In fact neither method achieves the 30 percent speedup we expect from the faster scalar speed of the X-MP. The envelope method, using RCM, is even worse. These methods depend on vectorized dot products in the factorization. A minor architectural change from the CRAY-1S to the X-MP

EXAMPLE 1  
 $N = 1,000, NZ = 2,750$

Algorithm	CRAY-1S		CRAY X-MP/24		Total 1S
	Factor	Solve	Factor	Solve	Total X-MP
RCM	0.309	0.010	0.291	0.008	1.07
OWD	0.439	0.018	0.389	0.016	1.13
RQT	0.412	0.024	0.361	0.020	1.14
AND	0.187	0.011	0.086	0.007	2.13
QMD	0.137	0.010	0.065	0.006	2.07

EXAMPLE 2  
 $N = 1,080, NZ = 14,630$

Algorithm	CRAY-1S		CRAY X-MP/24		Total 1S
	Factor	Solve	Factor	Solve	Total X-MP
RCM	1.509	0.019	1.436	0.013	1.05
OWD	1.514	0.019	1.439	0.014	1.05
RQT	3.011	0.037	2.538	0.027	1.19
AND	2.523	0.036	0.733	0.012	3.43
QMD	3.912	0.041	1.015	0.013	3.85

EXAMPLE 3  
 $N = 5,005, NZ = 15,028$

Algorithm	CRAY-1S		CRAY X-MP/24		Total 1S
	Factor	Solve	Factor	Solve	Total X-MP
RCM	2.648	0.055	2.615	0.043	1.02
OWD	4.134	0.101	3.516	0.081	1.18
RQT	3.774	0.124	3.229	0.103	1.17
AND	2.871	0.071	0.925	0.034	3.07
QMD	2.936	0.069	0.941	0.035	3.03

EXAMPLE 4  
 $N = 1,104, NZ = 1,341$

Algorithm	CRAY-1S		CRAY X-MP/24		Total 1S
	Factor	Solve	Factor	Solve	Total X-MP
RCM	0.119	0.008	0.111	0.007	1.08
OWD	0.214	0.020	0.186	0.017	1.16
RQT	0.182	0.024	0.156	0.020	1.17
AND	0.131	0.010	0.064	0.006	2.01
QMD	0.087	0.010	0.046	0.006	1.87

EXAMPLE 5  
 $N = 3,312, NZ = 11,025$

Algorithm	CRAY-1S		CRAY X-MP/24		Total 1S
	Factor	Solve	Factor	Solve	Total X-MP
RCM	1.069	0.031	1.060	0.025	1.01
OWD	2.239	0.068	1.926	0.055	1.16
RQT	1.974	0.080	1.693	0.063	1.17
AND	1.984	0.048	0.688	0.023	2.86
QMD	1.896	0.046	0.650	0.023	2.88

causes dot products to have a greater vector start-up time. This change has a considerable effect in practice. The combination of the two architectural changes causes a reversal in the desirability of different ordering algorithms.

A final point is worth making concerning a software issue. Our version of SPAR-SPAK had been modified with calls to computational kernels several years ago when it was installed on the CRAY-1S. These test examples were run again on the CRAY X-MP *without any code modifications*. Through computational kernels from a kernel library such as VectorPak the application programmer reaps the benefits of hardware improvements without concerning himself with the often subtle details of a new implementation. The success in using hardware gather/scatter in the context of sparse Gaussian elimination is one example validating the concept of computational kernels as a tool for combining portability and optimality on advanced architectures.

## REFERENCES

- [1] B. DEMBART AND K. NEVES, *Sparse triangular factorization on vector computers*, in Exploring Applications of Parallel Processing, Electric Power Research Institute, EL-566-QR, Palo Alto, CA, 1977, pp. 22-25.
- [2] D. S. DODSON AND J. G. LEWIS, *Proposed sparse extensions to the basic linear algebra subprograms*, SIGNUM Newsletter, 20 (1985), pp. 22-25.
- [3] I. S. DUFF, *The solution of sparse linear equations on the CRAY-1*, CRAY Channels, 4 (1982), pp. 4-9.
- [4] I. S. DUFF, R. GRIMES, J. LEWIS AND W. POOLE, *Sparse matrix test problems*, SIGNUM Newsletter, (1982), p. 22.
- [5] A. GEORGE AND J. LIU, *Computer Solution of Large Sparse Positive Definite Linear Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1982.
- [6] C. LAWSON, R. HANSON, D. KINCAID AND F. KROGH, *Basic linear algebra subprograms for Fortran usage*, ACM Trans. Math. Software, 5 (1978), pp. 308-323.
- [7] J. W. H. LIU, *Modification of the minimum degree algorithm by multiple elimination*, ACM Trans. Math. Software, 11 (1985), pp. 141-153.
- [8] K. W. NEVES, *The impact of changing architectures*, Report ETA-TR 26, Boeing Computer Services, 1985.
- [9] A. SHERMAN, *Linear Algebra for Reservoir Simulation Comparison Study of Numerical Algorithms*, J. S. Nolen Associates, 1984.
- [10] *VectorPak Subroutine Library User's Manual*, Document 20460-0501, Boeing Computer Services, 1986.
- [11] P. T. WOO AND J. M. LEVESQUE, *Benchmarking a sparse elimination routine on the Cyber 205 and the CRAY-1*, Proc. 6th SPE Symposium on Reservoir Simulation, 1982.