

# Triple Sparse Matrix Multiplication

Craig C. Douglas  
CS 521, Spring, 2002

January 22, 2002

## 1 Introduction

Almost all computational science involves modeling some process. Usually this means that a time dependent, nonlinear, coupled set of partial differential equations is in the process somewhere, either obviously or not.

Any collection of routines to solve partial differential equations or integral equations must support a variety of matrix storage formats. Discretizations of partial differential equations usually lead to large, sparse systems of equations. This will result in diagonal matrices, ones with a nearly constant number of nonzeros per row, or ones with a highly varying number of nonzeros per row. In all cases, the nonzero structure of the resulting matrix is usually nearly symmetric even when the matrix is not. On the other hand, integral equations and spectral discretizations of partial differential equations usually lead to dense systems of equations.

In the following section, we will investigate how to store a  $m \times n$  matrix  $M$ . In Table 1,  $mrows = m$  and  $mcols = n$  throughout.

Consider the following as an example of a  $5 \times 5$  general dense matrix  $M$ :

$$M = \begin{bmatrix} 11 & 0 & 13 & 0 & 0 \\ 21 & 22 & 0 & 0 & 25 \\ 0 & 0 & 33 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 51 & 52 & 0 & 0 & 55 \end{bmatrix} \quad (1)$$

Table 1: Matrix storage variables

Variable	Description
<i>mtype</i>	The matrix data type (see Table 2).
<i>mrows</i>	The number of rows in the matrix.
<i>mcols</i>	The number of columns in the matrix.
<i>cm</i>	The floating point coefficients of the matrix.
<i>iam</i>	The first integer associated matrix. This contains the starting location of a row in <i>jam</i> and <i>cm</i> .
<i>jam</i>	The second integer associated matrix. This contains the column indices.

Table 2: Data types

<i>mtype</i>	Data name	Floating point data description
1	float	single precision real
2	double	double precision real
3	complex	single precision complex
4	dcomplex	double precision complex

This example is unusual since it has a completely zero row. Normally, some process would filter the row out entirely.

There are many variants of sparse matrix storage: storage by rows, storage by columns, and stencil storage, to name a few. Each has its place.

We will concentrate on storage by rows, which is particularly useful for iterative solvers. By contrast, storage by columns is particularly useful for direct solvers like Cholesky or Gaussian elimination.

## 2 Storage by Rows

There are two common variants of storage by rows. The first is for general matrices. The second is for symmetric matrices. In the latter case, the lower triangular part of the matrix is *not* stored.

### 2.1 General Matrices

For a general, sparse matrix  $M$ , storage by rows uses three vectors to define the matrix:  $iam$ ,  $jam$ , and  $cm$ . Given the  $ne$  nonzero elements of  $M$ , the vectors are set up as follows:

- $cm$ , a floating point vector of length at least  $ne$ , contains the  $ne$  nonzero elements of the sparse matrix  $M$  stored contiguously. The rows of  $M$  are stored in ascending order. The elements of each row in  $M$  should be stored in *ascending order*.
- $iam$ , an integer vector of length at least  $m + 1$ , contains the relative starting position of each row of matrix  $M$  in vector  $cm$ . Hence, row  $i$  of  $M$  begins at  $cm(iam(i))$  and ends at  $cm(iam(i+1) - 1)$ . If row  $j$  is all zero (i.e., an empty row),  $iam(j) = iam(j + 1)$ . The last element,  $iam(m + 1)$ , indicates the position after the last element in vector  $cm$ , which is  $ne + 1$ .
- $jam$ , an integer vector of length at least  $ne$ , contains the corresponding column numbers of each nonzero element  $M_{ij}$  in matrix  $M$ .

Consider the example matrix  $M$  in (1) cast as a  $5 \times 5$  general sparse matrix. This can be stored as

Index	cm	iam	jam
1	11	1	1
2	13	3	3
3	21	6	1
4	22	7	2
5	25	7	5
6	33	10	3
7	51		1
8	52		2
9	55		5

## 2.2 Symmetric Matrices

For a symmetric matrix, about half of the storage can be eliminated. This is because  $M = M^T$ .

The storage is similar to the general case, but the definition of  $ne$  changes to include only the nonzero elements along the main diagonal and in the strictly upper triangular part of the matrix.

Consider the following as an example of a  $5 \times 5$  symmetric matrix  $S$ :

$$S = \begin{bmatrix} 11 & 0 & 13 & 0 & 0 \\ 0 & 22 & 0 & 0 & 25 \\ 13 & 0 & 33 & 0 & 0 \\ 25 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 55 \end{bmatrix} \quad (2)$$

This can be stored as

Index	cm	iam	jam
1	11	1	1
2	13	3	3
3	22	5	2
4	25	6	5
5	33	6	3
6	55	7	5

## 3 Matrix Multiplication

Computing  $C = AB$ , when  $A$  and  $B$  are dense is easy:

$$C_{ij} = \sum_{k=1}^n A_{ik} B_{ki}, \quad i = 1, \dots, m, \quad j = 1, \dots, n. \quad (3)$$

Implementing it efficiently is extremely difficult, however.

Normally all of the data passes through cache twice. Schemes have been developed to do (3) in blocked form so that data passes through cache only once. A cache aware scheme can run at almost the peak speed of a computer. A standard implementation runs at 10 – 20% of the peak speed.

For sparse matrices  $A$  and  $B$ , the nonzero structure of  $C$  has to be computed first. Then the numerical multiplication can be done efficiently. This is described in detail in [2] and a code can be found in [1].

A much more challenging problem is to efficiently compute  $C = RAP$ , where  $R$  is  $m_R \times n_R$ ,  $A$  is  $m_A \times n_A$ ,  $P$  is  $m_P \times n_P$ , and  $C$  is  $m_R \times n_P$ . An algorithm to determine the sparsity pattern can be found in [3].

The most common application of a triple matrix multiply is when  $A$  is a square matrix ( $m_A = n_A$ ),  $R$  has many more columns than rows ( $m_R < n_R$ ), and  $P$  is shaped like the transpose of  $R$  ( $m_P = n_R$  and  $n_P = n_R$ ).

## References

- [1] R. E. Bank and C. C. Douglas. SMMP code. See <http://www.ccs.uky.edu/~douglas/ccd-codes.html>.
- [2] R. E. Bank and C. C. Douglas. Sparse matrix multiplication package (SMMP). *Advances in Comput. Math.*, 1:127–137, 1993. See <http://www.ccs.uky.edu/~douglas/ccd-preprints.html>, pub. 34.
- [3] P. Wesseling. *An Introduction to Multigrid Methods*. John Wiley & Sons, Chichester, 1992. See <http://www.mgnet.org/mgnet-tuts.html>.