An Implementation of Quadtrees for Storing Choleski Factors

FLARE Group
Department of Computer Science
University of Wales, Swansea
Swansea SA2 8PP
Report No FLARE/S/10

28 Nov., 1992

Abstract

An implementation in Haskell of a Quadtree data structure is reported. The Quadtrees have been used for storing Choleski factors. They are experimentally compared with a conventional data structure, namely array of lists, used in the Generalized Envelope method. The effects of different matrix ordering have also be investigated.

1 Introduction

A Quadtree data structure designed for storing lower triangle Choleski factors has been implemented. Experimental results show that it out performs the array of lists data structure used in the implementation of Liu's generalized envelope method. The performance of these Quadtrees has been tested under four different orderings of system matrices. They include a bandwidth reduction, a profile reduction and a minimum degree ordering and a ordering algorithm tailored for the Quadtree structure.

2 Implementation of Quadtrees

For a lower triangle matrix like

$$L = \begin{bmatrix} L_1 & \\ E & L_2 \end{bmatrix},$$

a Quadtree structure is defined as

```
data TriMat a = TriM (TriMat a) (RectMat a) (TriMat a) | SingTM a
```

The first and third parameters following the constructor `TriM` accommodate the L_1 and L_2. The bottom-left rectangular submatrix is accommodated by the second parameter. The matrix is partitioned so that L_1 is always equal to one row larger than L_2. The constructor `SingTM` stands for an one-entry matrix.

For a rectangular matrix like

$$E = \begin{bmatrix} E_{11} & E_{12} \\ E_{21} & E_{22} \end{bmatrix},$$

the corresponding data structure is declared as

```
data RectMat a =
  RectM (RectMat a) (RectMat a) (RectMat a) (RectMat a)
  | ColRM a a | RowRM a a | SingRM a | ZeroM
```

The constructor `RectM` represents a matrix with four rectangular submatrices. The same partition strategy mentioned above is used here. Also, we need special constructors to cope with 2×1 and 1×2 matrices. `ColRM` and `RowRM` are thus provided. `SingRM` is for an one-entry matrix. `ZeroM` represents a **0** matrix.

Because we need to multiply the matrices with vectors, it is therefore convenient to represent vectors by (almost) balanced binary trees. Since we do not require direct random access to elements of these vectors this does not cause any extra access overheads.

3 Matrix Ordering

The distribution of nonzero entries in a sparse matrix can be affected by matrix orderings. This in turn can affect the performance of Quadtrees. To investigate the inference, four different orderings for system matrix Ks have be examined. Three of them are existing methods. They include implementations of a bandwidth reduction algorithm due to Grooms [1], a profile reduction algorithm due to Sloan [2], and a minimum degree ordering algorithm [3].

Another ordering algorithm is a modification to the Grooms'. It is designed to achieve such orderings so that the resulted Quadtrees have less branches. The idea behind the algorithm is very similar to that due to Grooms. In the Grooms' algorithm, nonzero entries are compared according to their distances to the matrix diagonal. The matrix is iteratively permuted so that the entry with the largest distance is moved closer to the diagonal. Instead of using a simple distance criteria, in this algorithm nonzero entries are compared according to the levels of blocks where they belong. Following the notations and partition scheme described earlier for matrices, the level sets, $S_i^L, i = 1, 2, .., $ of matrix subblocks are defined in a recursive manner:

1. $S_i^L = S_i^E, i = 1, 2.$
2. $S_i^L = S_{i-2}^E + S_{i-2}^{L1} + S_{i-2}^{L2}, i = 3, 4, ...$
3. $S_1^E = \{E_{21}\}.$
4. $S_2^E = \{E_{11}, E_{22}\}.$
5. $S_i^E = S_{i-2}^{E12}, i = 3, 4, ...$

The nonzero entry with the smallest block level (and largest distance to the diagonal if there is more than one such entry) is picked up and moved inside. This can be viewed as a Quadtree pruning process.

The reason why we used this algorithm is given bellow. Suppose we a matrix

$$\begin{bmatrix} A_1 & & & \\ B_1 & A_2 & & \\ C_1 & B_2 & A_3 & \\ & C_2 & B_3 & A_4 \end{bmatrix}.$$

where the submatrices correspond to a Quadtree partition. When this matrix is the result of a bandwidth reduction ordering, it is possible that those C_is are filled with a few nonzero entries while there are much room to be filled in A_is and B_is. If we can move those nonzero entries in C_is into A_is and B_is, we should be able to save some work for constructing binary trees while multiplying them with vectors.

4 Experimental Results

Three problems, `Problem-1`, `2` and `3`, described in report FLARE/S/9 have been used.

The execution time for the Quadtree implementation has been compared with some other data structures. They include using a tree of lists for the fixed and variable bandwidth schemes and a tree of trees of lists for the generalized envelope method [4]. To take into account the effect of the matrix structure, we apply various matrix ordering methods to the original K matrices. They are

A: the bandwidth reduction,

B: the profile reduction algorithm, and

C: the minimum degree ordering algorithm.

Table 1 compares the average execution time for the solution of equation (3) when Quadtrees and trees of lists are used. It is obvious that the Quadtree is much better than the tree of lists for both orderings.

Table 2 compares the execution time and size of the Quadtrees and the tree of trees of lists data structure used in the generalized envelope method (using ordering scheme C). We have used the generalized envelope method because it eliminates the storage of all zero entries in a Choleski matrix factor [5]. Therefore, it eliminates unnecessary computations arising from operating on zero entries, of which there are many, and minimizes the number of numerical computations. In that sense, it is the best among all row oriented storage schemes. The size of a tree is taken as the total number of internal nodes plus the total size of the leaves in the trees (the size for a list leaf is its length). It is seen that a Quadtree is roughly twice the size of the equivalent tree of trees of lists. As far as the execution speed is concerned, the Quadtree is at least as good as its counterpart for Problems 1 and 2. We believe, for Problem 3, the anomalous performance of the Quadtree data structure in comparison with the tree of trees of lists is due to the large space requirement. Comparing the Quadtree results in Table 1 and Table 2, it is seen that the Quadtree is not very sensitive to the matrix structure. Contrasting this insensitivity and the necessity of applying the minimum degree ordering, which is time consuming, for the generalized envelope method, we conclude that the Quadtree is generally more acceptable.

We have also tried the aboved mentioned ordering scheme (modification to the Grooms' scheme). For these test problems, the achieved results are comparable to those given in Table 1 and Table 2.

References

[1] H.R. Grooms. Algorithm for matrix bandwidth reduction. *Proc. ASCE, J. Struct. Div. ST1, paper 8636*, 1972.

[2] S. W. Sloan. An Algorithm for Profile and Wavefront Reduction of Sparse Matrices. *Int. Journal Num. Meth. Eng.*, 23:239–251, 1986.

[3] I.S. Duff, A.M. Erisman, and J.K. Reid. *Direct Methods for Sparse Matrices.* Clarendon Press. Oxford, 1986.

[4] P.W. Grant, J.A. Sharp, M.F. Webster, and X. Zhang. A Haskell implementation of a Generalized Envelope method for sparse matrix factorization. In *Proc. ATABLE-92*, Montreal, June 1992.

[5] Joseph W.H. Liu. A generalized envelope method for sparse factorization by rows. *ACM Trans. on Math. Software*, 17:112–129, March 1991.

Ordering	Problem	Time (sec)	
		Quadtree	tree of lists
A	1	1.0	4.1
	2	2.4	10.8
	3	8.1	42.1
B	1	1.6	3.8
	2	2.5	10.7
	3	6.0	39.2

Table 1: Time comparison of Quadtrees and trees of lists

	Problem	Quadtree	tree of trees of lists
Time (sec)	1	1.3	2.1
	2	2.1	3.9
	3	10.9	8.7
tree size	1	3027	1504
	2	5212	2687
	3	14950	7205

Table 2: Time and tree size comparison of Quadtrees and the generalized envelope method