

# B2SES

## A High-Performance Sparse Solver for B2000\*

Mathias Doreille  
*SMR Engineering & Development*

November 24, 2000

### Abstract

This paper presents a new sparse solver for symmetric positive definite linear system for B2000. After an overview of the sparse solver we present the new module B2SES that implements both an in-core supernodal right-looking sparse Cholesky factorisation and an out-of-core supernodal multifrontal sparse Cholesky factorisation. Finally, some experiments are presented which demonstrate the large improvement of this solver in term of execution time and memory requirements compared to the old B2000 skyline solver B2ES.

### Introduction

In structural mechanics, implicit finite element methods are most commonly used for static linear or non-linear analysis. Especially in the case of shell problems and composite materials, the condition of the stiffness matrix is bad, thus severely limiting the use of iterative solvers in computational structural mechanics. Instead, the resulting linear system has to be solved by means of direct methods. Standard  $LL^t$  decomposition requires the stiffness matrix to be stored in skyline form. Solving large problems becomes very time and resource consuming. Therefore, a new direct solver using a compressed storage scheme and sparse Cholesky factorisation has been developed and implemented in B2000.

In the first section of this paper, we will present a short overview of the state of the art in direct solver techniques for sparse symmetric linear systems. In section 2, we will present the implementation of a new sparse direct solver in B2000, the module B2SES. Section 3 is devoted to experiments, with comparisons between the old B2000 solver B2ES and different configurations of the new solver B2SES.

---

\*Paper presented at the 3rd B2000 workshop, 27-28 November 2000, University of Twente, the Netherlands.

# 1 Sparse Cholesky factorisation

The sparse Cholesky factorisation algorithm is an adaptation of the Gauss elimination method for the symmetric case, in which only operations on nonzero entries are performed. This is the major difference from the skyline Cholesky factorisation algorithm that only removes a part of the operations on null entries.

The basic sparse Cholesky factorisation algorithm is rather straightforward. As an illustration, we present in figure 1 the right looking version of it, described at the level of element operations.  $Col(k)$  denotes the set of row indices of subdiagonal non-zero elements of column  $k$ . However, an efficient implementation of the sparse Cholesky factorisation requires many improvements and the use of complex optimisation techniques.

**Require:** sparse symmetric positive defined matrix  $A$  of size  $n$

**Ensure:**  $LL^T = A$

```
1:  $L$  is initialized with  $A$ 
2: for  $k = 1$  to  $n$  do
3:    $L_{k,k} \leftarrow \sqrt{L_{k,k}}$ 
4:   for  $i \in col(k)$  do
5:      $L_{i,k} \leftarrow \frac{L_{i,k}}{L_{k,k}}$ 
6:   end for
7:   for  $j \in col(k)$  do
8:     for  $i \in col(k)$  and  $i \leq j$  do
9:        $L_{i,j} \leftarrow L_{i,j} - L_{i,k}L_{j,k}^t$ 
10:    end for
11:  end for
12: end for
```

Figure 1: Sparse right-looking Cholesky factorisation algorithm.

## 1.1 Reordering and symbolic factorisation

For optimisation, two preliminary steps must be performed before the actual numerical factorisation. In the first step, called reordering, a permutation of the initial matrix is computed to reduce the fill-in occurring during factorisation. This is crucial, since this determines the storage requirement and the number of operations required to compute the factored matrix. The computation of an optimal nodes permutation for reordering a given matrix with the goal to minimize fill-in during factorisation is known to be NP difficult<sup>1</sup> [12]. However, different heuristics have been proposed. Two different schemes are used to compute a reordering: Minimum degree [7] and nested dissection [10]. Both of them generally provide good reordering. We will give some comparisons of these two schemes in the experiments.

In the second step, called symbolic factorisation, the structure<sup>2</sup> of the factored

---

<sup>1</sup>Class of problem that “can not” be solved in polynomial time.

<sup>2</sup>The structure of a sparse vector or matrix refers to the position of non zero elements in this vector or matrix.

matrix is computed. This is also crucial for efficiency, since storage requirements are solved after this stage. Thus, dynamic memory management is eliminated during the numerical factorisation. Many optimisations [8] techniques are proposed to compute symbolic factorisation.

These two steps, reordering and symbolic factorisation, are computed using only information on the matrix structure without taking into account their numerical value. Thus, if many linear systems with the same matrix structure need to be factorised, these preliminary steps are only computed once. This can be the case for instance in case of nonlinear analysis with a Newton-Raphson technique, or in case of multiple right-hand sides.

We note that these two preliminary steps are possible only for sparse symmetric definite positive linear systems on which pivoting is not necessary during numerical factorisation. For sparse undefined or unsymmetric linear systems, row or column pivoting must be used for numerical stability reasons.

## 1.2 Numerical Factorisation

After these preliminary steps, the numerical factorisation can be performed. This third stage is the most time consuming phase and several different ways are proposed to efficiently organize the computations. The principle optimisation for this step is to transform the nested loop of algorithm 1 to intensively use BLAS 3 [3] subroutines. This is crucial for the effectiveness of linear algebra programs on modern processors which have greater speed in performing arithmetic operations compared to their memory accesses. As BLAS 3 routines operate at block level, blocks may be kept in the fast memory (cache) while performing several operations using the same matrix elements (generally  $n^3$  operations for  $n^2$  elements). This way the floating point unit of the processor can be fed at full speed.

### 1.2.1 Supernodal technique

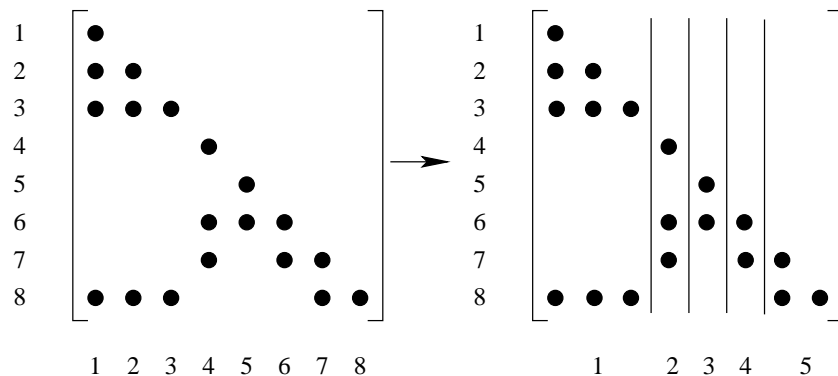


Figure 2: Sparse matrix structure of a factor  $L$  obtained after symbolic factorisation and the associated supernode.

A supernode is a group of columns than can be treated as one computational unit in the course of a sparse Cholesky factorisation. For this, all columns in a group

**Require:** sparse symmetric positive defined matrix  $A$  of size  $n$ , partitioned in  $N$  supernodes noted  $L_j, 1 \leq j \leq q$

**Ensure:**  $LL^T = A$

- 1: **for**  $k = 1$  to  $N$  **do**
- 2:   dense Cholesky factorisation of the bloc  $L_k$
- 3:   **for**  $j \in Struct(L_{*,k})$  **do**
- 4:     updating of bloc  $L_j$  with the bloc  $L_k$
- 5:   **end for**
- 6: **end for**

Figure 3: Sparse supernodal right-looking Cholesky factorisation algorithm.

must have the same structure. The figure 2 shows an example of the structure of a factorised matrix  $L$  (structure is known after the symbolic factorisation) and the associated supernodes.

Figure 3 shows the supernodal version of the right-looking algorithm we showed in figure 1. In this algorithm, operations at line 2 are performed with BLAS 3 routine `tpotrf` and `dtrsm` and operation at line 4 is performed with the BLAS 3 routine `dgemm` (matrix-matrix product).

Another technique, proposed by Ashcraft and Grimes [2], to improve the performance of supernode sparse Cholesky factorisation is to allow the grouping of several supernodes into a greater supernode, with the sacrifice of treating some zeros as nonzeros, such that all columns of the new supernode have the same row structure. By increasing supernode size, this technique improves the performance of the BLAS 3 operations. Figure 1.2.1 shows an example of supernode amalgamation that amalgam three supernodes (3,4 and 5) in one supernode by treating two zeros as nonzero.

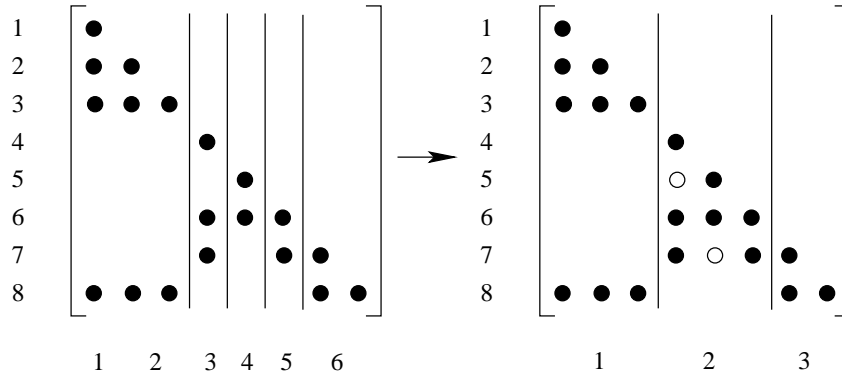


Figure 4: Supernodal amalgamation by adding zero elements (white circle) in the structure of the factorised matrix

### 1.2.2 Out-of-core sparse multifrontal Cholesky factorisation

There are two major different ways to organize the computation of sparse numerical factorisation, all based on a Gaussian elimination process: Column oriented algo-

rithms, like the right-looking algorithm already presented in figure 1, and frontal oriented algorithms, like the multifrontal [11] algorithm.

Supernodal column oriented algorithms are generally more efficient than supernodal multifrontal algorithms, but the latter can be transformed to an efficient out-of-core algorithm.

## 2 The new B2000 module B2SES

The methods for sparse Cholesky factorisation described above are implemented in the module B2SES, except for the nested dissection reordering algorithm that can be performed by calling an external library. This way, B2SES can use two different reordering algorithms. An approximate minimum degree algorithm based on a quotient graph with many optimisations such as element absorption, mass elimination, supernodes and external degree [1]. And B2SES can also be linked with the Metis library [9] that provides a multilevel nested dissection reordering algorithm.

The numerical factorisation can also be computed with two different methods: An in-core supernodal right-looking algorithm for linear systems that can be solved in-core, and an out-of-core supernodal multifrontal algorithm for solving very large linear systems out-of-core. These two algorithms perform a  $LDL^t$  factorisation on symmetric non-positive definite linear systems. However, since row and column pivoting are not performed, numerical stability is not assured in this case.

To compute the linear system coming from implicit finite element problems, two other stages are also present in B2SES. The first, performed before the sparse Cholesky factorisation, computes the matrix structure of the linear system (i.e. the nonzero row indices of the sparse matrix) from the stiffness matrix structure of each element. The second, performed after symbolic factorisation and before numerical factorisation, computes the value of the reordered linear system by assembly of the stiffness matrix of each element. A Lagrange multiplier technique is also used to take into account linear dependency.

The module is implemented mainly in C++, using the STL and BLAS 3 routines whenever possible. The MemCom library has been used not only to access the B2000 database but also for the out-of-core mechanism. For this, each out-of-core array is mapped on a buffered (paged) database file.

## 3 Experiments

We have tested the module B2SES on three different finite element problems. A structured shell problem called “Shell”, a plate with  $200 \times 200$  “Q4” elements. Each element has 4 nodes with 6 degrees of freedom. The second problem is a structured volume problem called “Beam”, a beam with  $20 \times 40 \times 50$  “HE8” elements. Each element has 8 nodes with 3 degrees of freedom. The last problem is an unstructured volume problem containing three different volume elements, “TE10”, “WE15” and “HE20”, totaling 157170 degrees of freedom. A snapshot of the visualization of this problem with baspl++ is shown in figure 5.

The experiments have been run on an Alpha bi-processor machine with 2 Gbyte of memory running Compaq Tru64 UNIX. Its peak performance per processor for

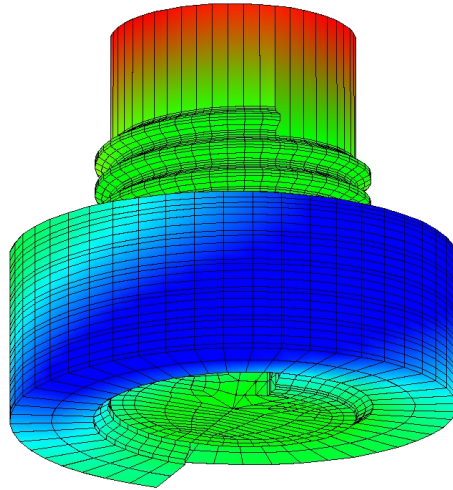


Figure 5: Stress calculation of a bolt.

the BLAS 3 routine `dgemm`<sup>3</sup> provided by Compaq is about 1180 Mflop/s and the performance for the routine `dpotrf`<sup>4</sup> is about 680 Mflop/s.

### 3.1 Reordering algorithms

Table 1 shows the number of floating point operations and storage size acquired to factorise the linear system. We compare the old skyline solver making use of the Reverse Cuthill-McKee reordering algorithm and the new solver with the two reordering algorithms supported by B2SES, i.e. approximate minimum degree (AMD) and nested dissection using the Metis library. The second row of the table shows the nonzero element in the initial linear system. For each reordering algorithms we give the number of nonzeros in the factored matrix  $L$  and the number of floating point operations required to compute this factored matrix.

This result demonstrates the large improvement in storage size, particularly for unstructured meshes for the sparse solver. We can also show that the nested dissection algorithm is better than the approximate minimum degree algorithm for this type of problems.

Table 2 shows the corresponding computing times required to compute the reordering using approximate minimum degree and nested dissection. For all problem, AMD is faster than nested dissection.

The following experiments will now exclusively make use of the nested dissection reordering algorithm which is the most efficient for the problems used.

### 3.2 Numerical factorisation and backward substitution

Table 3 shows the different execution times of the skyline solver B2ES, the new solver B2SES with in-core right-looking Cholesky factorisation and the new solver

---

<sup>3</sup>Matrix matrix product on 64 bits float.

<sup>4</sup>Dense Cholesky factorisation on 64 bits float.

		Shell	Beam	Bolt
Nonzero in initial matrix		4524315	4524510	11815701
Skyline, RCM:	Nonzero in $L$	2089	2903	8140
Sparse, AMD:	Nonzero in $L$	443	1894	2716
	Mflop for $L$	44510	71742	?
Sparse nested dissection:	Nonzero in $L$	306	828	1390
	Mflop for $L$	27238	80128	102880

Table 1: Factorised matrix storage size in floats (8 bytes) and number of float point operation required to compute this factored matrix  $L$  in Mflop for different reordering algorithms.

	Shell	Beam	Bolt
AMD	1.56s	2.07s	5.48s
Nested dissection	1.85s	3.65s	8.20s

Table 2: Execution times in seconds for computing a reordering with the approximate minimum degree algorithm and the nested dissection algorithm.

B2SES with out-of-core multifrontal Cholesky factorisation. For a comparison with the skyline solver that only runs with relatively small problems, we added a new problem “small Shell” that consists of  $100 \times 100$  shell elements. A “X” denotes the factorisation can not be performed because of the memory being limited to 2 Gbyte on the machine. This result demonstrates the large improvement in execution times of the new solver compared with the original skyline solver B2ES. Moreover, we can show that for relatively small problems which can be solved in-core, the performance in Mflop/s of the new sparse Cholesky solver approach the performance peak of the Alpha processor rated at 1180 Mflop/s.

		small Shell	Shell	Beam	Bolt
Skyline:	Time	1120.0s	X	X	X
In-core right-looking:	Time	3.63s	31.8s	271.4	X
	Mflop/s	914.0	856.5	295.2	X
Out-of-core multifrontal:	Time	10.05s	59.4s	409.7s	991.6s
	Mflop/s	330.1	458.5	195.6	103.8

Table 3: Execution times and performance in Mflop/s of numerical factorisation with skyline, in-core right looking and out-of-core multifrontal Cholesky factorisation for various problems.

Table 4 details the execution time of each step of the sparse solver. We can see that the two preliminary stages of reordering and symbolic factorisation are negligible compared to the numerical factorisation.

	Shell	Beam	Bolt
Building of the matrix structure	1.6s	2.1s	3.7s
Reordering with METIS	1.8s	3.6s	7.9s
Symbolic factorisation	0.4s	0.7s	1.3s
Assembly of the matrix	2.8s	3.5s	7.1s
Numerical factorisation	57.9s	408.2s	988.2s
Backward substitution	5.7s	19.0s	39.4s

Table 4: Execution times of each step of out-of-core sparse supernodal multifrontal Cholesky factorisation.

### 3.3 Comparison with iterative solver

The execution time comparison of a direct solver with an iterative solver is not significant since for iterative solvers, execution time only depends on the required precision. However, this comparison can still be of interest.

For this comparison we use the Conjugate Gradient method which is the most appropriate for symmetric positive definite system. We also used the SSOR method to precondition the system. We implemented these algorithms in B2000 using the SPARSE BLAS library [6] which provides the sparse matrix dense vector product operations.

The iterative solver does not converge for the problems described above. However, we can detect a “little” convergence for a small diffusion problem with  $100 \times 100$  Q4 elements and 9800 degrees of freedom. This problem is solved with the B2SES direct solver in 0.22 seconds. Table 5 shows the results obtained with the iterative solver. We give execution time and residue obtained at different iteration steps. The best residue we can obtain is 0.056. This shows that this iterative solver can not be used for this type of problems.

Number of iteration	5	10	20	40	80
Execution times	0.017s	0.033s	0.083	0.133s	0.283s
Residue	0.287	0.121	0.076	0.057	0.056

Table 5: Iterative resolution of a diffusion problem with preconditioned SSOR Conjugate Gradient method for various problems.

### 3.4 Parallel execution using a parallel BLAS library

A simple approach for executing in parallel a code that intensively uses the BLAS library on SMP machines consists of making use of the parallel BLAS library usually provided by the manufacturer of the platform. Table 7 shows the performance of some tests for the in-core and the out-of-core solver on the bi-processor Alpha machine with the Compaq parallel BLAS library `dxmlp`. The best obtained speedup is 1.62 for the “Beam” problem using the in-core solver. However, no speedup is obtained with the out-of-core solver.

	Shell	Beam
Sequential BLAS	31.9s	271.6s
Parallel BLAS	27.9s	167.7s
Speedup	1.14	1.62

Table 6: Execution times of the in-core right looking numerical factorisation with sequential and parallel BLAS library for various problems.

	Shell	Beam	Bolt
Sequential BLAS	59.4s	411.3s	1044.9s
Parallel BLAS	67.6s	439.7s	1056.5s
Speedup	0.88	0.93	0.99

Table 7: Execution times of out-of-core multifrontal numerical factorisation with sequential and parallel BLAS library for various problems.

For a more salable approach, an experimental distributed sparse Cholesky solver with two-dimensional bloc partitioning [4] has been integrated in B2000. This solver, that uses MPI and BLAS libraries, has been tested on different distributed machines (IBM SP1, Cray T3D). Experiments realized with sparse matrix of the Harwell-Boeing collection [5] have shown a maximum speedup of 7 on 32 processors of an IBM SP1 and a maximum speedup of 20 on 32 processors of a CRAY T3D.

## 4 Conclusion

We have presented a new sparse solver for symmetric positive definite systems for B2000 that greatly improves the performance execution time and the memory requirements. This solver intensively use BLAS 3 library and many other optimisations. Two versions of this solver are proposed: An in-core solver using supernodal sparse right-looking Cholesky factorisation. This version provides the better performance for problems than can be solved in core. And an out-of core solver using supernodal multifrontal Cholesky factorisation. The experiments have shown that these solvers can reduce the time factorisation by a factor of 300 compared to the old skyline solver. That also permits to solve some problems with a size up to 157000 degree of freedom and a sparse matrix with up to 12.000.000 nonzero elements on a machine with 2Gbyte of memory.

## References

- [1] P.R. Amestoy, T.A. Davis, and I.S. Duff. An Approximate Minimum Degree Ordering Algorithm. *SIAM J.Matrix Anal.Appl.*, 17(4):886–905, October 1996.
- [2] C. Ashcraft and R. Grimes. The Influence of Relaxed Supernode Partitions on the Multifrontal Method. *ACM Trans.Math.Soft.*, 15(4):291–309, December 1989.

- [3] J.J. Dongarra, J. Du Croz, S. Hammarling, and I. Duff. A Set of Level-3 Basic Linear Algebra Subprograms. *ACM Trans.Math.Software*, 16:1–28, 1990.
- [4] M. Doreille, B. Dumitrescu, J.-L. Roch, and D. Trystram. Two-dimensional block partitionings for the parallel sparse Cholesky factorization. *Numerical Algorithms*, 16(1):17, February 1998.
- [5] Iain Duff, Roger G. Grimes, and John G. Lewis. Users' guide for the Harwell-Boeing sparse matrix collection (Release I). Technical Report TR/PA/92/86, CERFACS, October 1992.
- [6] Iain Duff, Michele Marrone, Giuseppe Radicati, and Carlo Vittoli. Level 3 basic linear algebra subprograms for sparse matrices: a user level interface. *ACM Transactions on Mathematical Software*, 23(3):379–401, September 1997.
- [7] A. George and J.W.H. Liu. The Evolution of the Minimum Degree Ordering Algorithm. *SIAM Review*, 31(1):1–19, March 1989.
- [8] Alan George and Joseph W.H. Liu. An optimal algorithm for symbolic factorization of symmetric matrices. *SIAM Journal of Computing*, 9(3):583–593, August 1980.
- [9] G. Karypis and V. Kumar. METIS – Unstructured Graph Partitioning and Sparse Matrix Ordering System, version 2.0. Technical report, Department of Computer Science, University of Minnesota, Minneapolis, 1995.
- [10] R.J. Lipton, D. J. Rose, and Tarjan R. E. Generalized nested dissection. *SIAM Journal on Numerical Analysis*, 16:346–358, 1979.
- [11] J.W.H. Liu. The Multifrontal Method for Sparse Matrix Solution: Theory and Practice. *SIAM Review*, 34(1):82–109, March 1992.
- [12] M. Yannakakis. Computing the minimum fill-in is np-complete. *SIAM Journal on Algebraic and Discrete Methods*, 2(1):77–79, March 1981.