Parallel multifrontal method with *out-of-core* techniques

Abdou Guermouche[⋆]

LaBRI, Bordeaux, France

Abstract. The memory usage of sparse direct solvers can be the bottleneck to solve large-scale problems. We describe the ongoing work on the implementation of an *out-of-core* extension to a parallel multifrontal solver (`MUMPS`). We show that larger problems can be solved on limited-memory machines with reasonable performance, and we illustrate the behaviour of both the parallel *out-of-core* factorization and the parallel *out-of-core* solution steps. We finally give some words on our future work in the field of sparse parallel out-of-core solvers.

1 Introduction

The solution of sparse systems of linear equations is a central kernel in many simulation applications. Because of their robustness and performance, direct methods can be preferred to iterative methods. In direct methods, the solution of a system of equations $Ax = b$ is generally decomposed into three steps: (i) an analysis step, that considers only the pattern of the matrix, and builds the necessary data structures for numerical computations; (ii) a numerical factorization step, building the sparse factors (e.g., L and U if we consider an unsymmetric LU factorization); and (iii) a solution step, consisting of a forward elimination (solve $Ly = b$ for y) and a backward substitution (solve $Ux = y$ for x). For large sparse problems, direct approaches often require a large amount of memory, that can be larger than the memory available on the target platform (cluster, high performance computer, ...). In order to solve increasingly large problems, *out-of-core* approaches are then necessary, where disk is used to store data that cannot fit in physical main memory.

Although several authors have worked on sequential or shared-memory *out-of-core* solvers [1, 5, 10], we do not know of any *out-of-core* direct solver for distributed-memory machines. In this work, we aim at extending a parallel multifrontal solver (`MUMPS`, for MUltifrontal Massively Parallel Solver, see [3]), in order to enable the solution of larger problems, thanks to *out-of-core* approaches. Recent contributions by [8] and [9] for uniprocessor approaches pointed out that multifrontal methods may not fit well an *out-of-core* context because large dense matrices have to be processed, that can represent a bottleneck for memory; therefore, they prefer left-looking approaches (or switching left-looking approaches). However, in a parallel context, increasing the number of processors can help keeping such large frontal matrices in-core.

We present in this paper the current state of our work on an *out-of-core* extension of the parallel mutifrontal solver `MUMPS` and give some words about our ongoing and future work in the field.

[⋆] All members of the current `MUMPS` development team have contributed to the work described in the paper.

2 Memory management in a parallel multifrontal method

In multifrontal methods, the task dependencies are represented by a so-called assembly tree [4, 6], that is processed from bottom to top during the factorization. At each node of the tree is associated a so-called *frontal matrix*, or *front*, and a task consisting in the partial factorization of the frontal matrix. The partial factorization produces a Schur complement, or *contribution block*, which will be used to update the frontal matrix of the parent node (see [2], for example, for more details). This leads to three areas of storage, one for the factors, one for the contribution blocks, and another one for the current frontal matrix [2]. The active memory (as opposed to the memory for the factors) then corresponds to the sum of the contribution blocks memory (or stack memory) and the memory for the current active matrix. During the factorization process, the memory required for the factors always grows while the stack memory that contains the contribution blocks varies: when the partial factorization of a frontal matrix is performed, a contribution block is stacked which increases the size of the stack; on the other hand, when the frontal matrix of a parent is formed and assembled, the contribution blocks of the children nodes can be discarded and the size of the stack decreases[1].

3 Out-of-core multifrontal approach

In the multifrontal method, the factors produced during the factorization step are not reused before the solution step. It then seems natural to first focus on writing *them* to disk. Thus, a first *out-of-core* scheme is to write factors to disk as soon as they are computed. In this context, we designed several mechanisms based on different I/O schemes. The synchronous I/O scheme is directly based on the standard I/O subroutines (either *fread/fwrite* or *read/write*). In the other hand, the asynchronous I/O scheme has been designed by associating to each process of our application with an I/O thread which is in charge of doing all I/O operations. The I/O thread is designed over the standard POSIX thread library (pthread library). Together with these two main schemes, we designed a buffered I/O mechanism (at the application level) that can work with both schemes.

3.1 Experiments

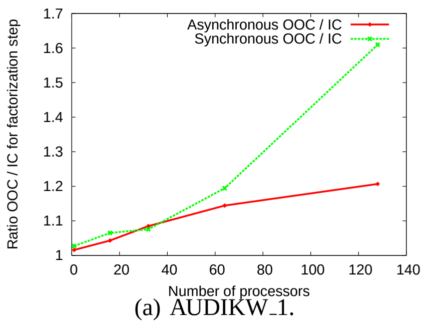
In order to study the impact of the proposed mechanisms, we experimented them on several problems extracted from either the PARASOL collection[2] or given from other sources. The tests have been performed on the IBM SP system of IDRIS[3] composed of several nodes of either 4 processors at 1.7 GHz or 32 processors at 1.3 GHz. On this machine, we have used from 1 to 128 processors with the following memory constraints: we can access 1.3 GB per processor when asking for more than 128 processors, 3.5 GB per processor for 17-64 processors, 4 GB for 2-16 processors, and 16 GB on 1 processor.

We report in Figure 1(a), a comparison between the time needed for factorizing the matrix when using the synchronous I/O scheme and the asynchronous buffered scheme (the results are normalized with the time needed for the *in-core* factorization). Note that for the CONESHL_MOD the results have been obtained using the CRAY XD1 system at CERFACS which has local disks per processor.

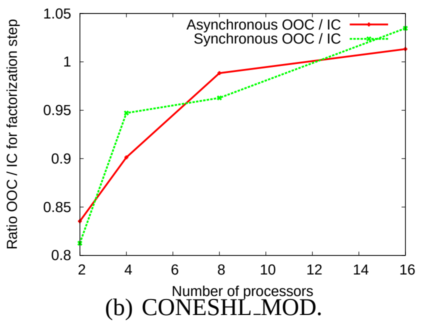
[1] In parallel, the contribution blocks management may differ from a pure stack mechanism.

[2] `http://www.parallab.uib.no/parasol`

[3] Institut du Développement et des Ressources en Informatique Scientifique



(a) AUDIKW_1.



(b) CONESHL_MOD.

Fig. 1. Execution times (normalized with respect to the *in-core* case) of the synchronous I/O scheme and asynchronous bufferized I/O scheme (`METIS` [7]) is used as reordering technique).

First, we have been able to observe that for a small number of processors we use significantly less memory with the *out-of-core* approach: the total memory peak is replaced by the active memory peak, with improvement ratios going up to 80%. Thus the factorization can be achieved on limited-memory machines.

We now focus on performance issues and report in Figure 1 a comparative study of the *in-core* case, the synchronous *out-of-core* scheme and the asynchronous buffered scheme.

Note that for the buffered case, the size of the I/O buffer is set to twice the size of the largest factor block (to have a double buffer mechanism). As we can see, the performance of the *out-of-core* schemes is indeed close to the *in-core*.

Concerning the parallel case, we observe that with the increase of the number of processors, the gap between the *in-core* and the *out-of-core* cases increases. The main reason is the performance degradation of the I/O with the number of processors due to the use of the `GPFS` file system on the IBM machine. In order to avoid this problem, we have experimented with the CONESHL_MOD problem on a machine with local disks. In this case, we do not have such a performance degradation, as shown in Figure 1(b); on the contrary, the *out-of-core* schemes perform as well or even better than the *in-core* one (cache effects resulting from freeing the factors from main memory and using always the same memory area for active frontal matrices). Finally, concerning the comparison of the *out-of-core* schemes, we can see that the asynchronous buffered approach performs better than the synchronous one. This illustrates that when the overlapping becomes critical, the asynchronous buffered approach is more appropriate than the synchronous one.

4 Out-of-core solution step

Concerning the solution phase, the size of the memory will generally not be large enough to hold all the factors. Thus, factors have to be read from disk, and the I/O involved increase significantly the time for solution. Note that we use a basic demand-driven scheme, relying on the synchronous low-level I/O mechanisms from Section 3. We have observed that the performance of the *out-of-core* solution step is often more than 10 times slower than the *in-core* case. Although disk contention might be an issue on our main target platform in the parallel case, the performance of the solution phase should not be neglected; it becomes critical in an *out-of-core* context and prefetching techniques in close relation with scheduling issues have to be studied.

5 Future work

We presented in this paper a first implementation of an *out-of-core* extension of the parallel multifrontal solver `MUMPS`. The selected approach was to drop factors from memory as soon as they are computed and to overlap the I/O operations as much as possible with computations. We illustrated the good behaviour of this approach on a small number of processors and its limitations on larger ones, while first experiments on machines with local I/O showed no significant I/O overhead during the factorization.
One key point that must be studied is the design of efficient *out-of-core* stack memory management. In this context, the contribution blocks can be considered as read-once/write-once data accessed with a near-to-stack mechanism (for the parallel case the accesses are more irregular). With asynchronous I/O, prefetching algorithms have to be designed. In addition, the number of contribution blocks (for the parallel case) that a processor has in memory is closely related to the scheduling decisions made; both the static and dynamic aspects of scheduling could limit the I/O volume that each processor has to perform and drive some dynamic decisions with the data that are available in memory (for example, give a priority to tasks that depend on/consume contribution blocks already in memory).
Concerning the solution step, we presented a preliminary study of a basic *out-of-core* scheme and identified that much work remains to be done in order to prefetch the factors adequately. In addition, in the parallel context, it is important to improve the interaction between the scheduling algorithms and the prefetching techniques used.

References

1. The BCSLIB Mathematical/Statistical Library. http://www.boeing.com/phantom/bcslib/.
2. P. R. Amestoy and I. S. Duff. Memory management issues in sparse multifrontal methods on multiprocessors. *Int. J. of Supercomputer Applics.*, 7:64–82, 1993.
3. P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.
4. C. Ashcraft, R. G. Grimes, J. G. Lewis, B. W. Peyton, and H. D. Simon. Progress in sparse matrix methods for large linear systems on vector computers. *Int. Journal of Supercomputer Applications*, 1(4):10–30, 1987.
5. F. Dobrian and A. Pothen. Oblio: a sparse direct solver library for serial and parallel computations. Technical report, Old Dominion University, 2000.
6. I. S. Duff and J. K. Reid. The multifrontal solution of indefinite sparse symmetric linear systems. *ACM Transactions on Mathematical Software*, 9:302–325, 1983.
7. G. Karypis and V. Kumar. METIS – *A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices – Version 4.0*. University of Minnesota, September 1998.
8. E. Rothberg and R. Schreiber. Efficient methods for out-of-core sparse Cholesky factorization. *SIAM Journal on Scientific Computing*, 21(1):129–144, 1999.
9. Vladimir Rotkin and Sivan Toledo. The design and implementation of a new out-of-core sparse Cholesky factorization method. *ACM Trans. Math. Softw.*, 30(1):19–46, 2004.
10. S. Toledo. Taucs: A library of sparse linear solvers, version 2.2, 2003. Available online at http://www.tau.ac.il/˜stoledo/taucs/.