A Preliminary Analysis of the Out-of-Core Solution Phase of a Parallel Multifrontal Approach

P. Amestoy [*] I.S. Duff [†] A. Guermouche [‡] Tz. Slavova [§]

2nd June 2006

We consider the solution of sparse linear systems of equations in a limited memory environment. A preliminary out-of core (**OOC**) version of a sparse multifrontal code (MUMPS, MUltifrontal Massively Parallel Solver) has been developed (this is part of a collaboration with members of the INRIA project GRAAL). In this OOC version the complete matrix of factors is written to disk during the factorization phase as a sequence of blocks (so called **factor blocks**) during the processing of the dependency graph (so called **elimination tree**). During the subsequent phase (forward and backward solutions, the so called **solve phase**) we then have to load the factor blocks from the local disks of the computer to the main memory. In this context, the cost of the solution phase can become the dominant phase of the complete solution process. When the solution phase has to be performed for many right-hand sides (simultaneously or not) then it is even more critical. In this talk, we focus on the solution phase and show the limitations of an approach based on automatic system I/O caching mechanisms, so called **system based approach**. Then we show how user buffers can be introduced to improve the behaviour of the solve phase. We conclude by illustrating the influence of task scheduling on the uniprocessor and multiprocessor solution phase. Our work differs and extends the work of Rothberg and Schreiber (1999) and Rotkin and Toledo (2004) because firstly we consider a parallel out-of-core context and secondly we focus on the performance of the solve phase. Two matrices will be used for the experiments. Grid 300-100-10 corresponds to an 11-point discretization of the Laplacian operator on a three-dimensional grid problem of size $300 \times 100 \times 10$ (factor size: 748 MB). Qimonda07 from Qimonda AG company is a large matrix from circuit simulation of order 8,613,291 (factor size: 2534 MB). All runs have been performed on the multiprocessor Cray XD1 located at CERFACS (58 nodes with 2 processors per node and 4 Gbytes per node). Each node is equipped with an `IDE` disk managed by the `reiserfs` file system of maximum bandwidth for a read operation close to 16 MB/sec.

Limitation of the system based approach

One simple way to implement the OOC solution phase is to let the system handle intermediate buffers/caches, prefetching (look-ahead read mechanisms) and blocking to access factor blocks. The implementation of the OOC solve phase is in this case very easy. We illustrate in the following table the potential and limitations of such an approach.

Strategy	Factor Time (sec)	Solve Fwd (sec)	Solve Bwd (sec)	Solve Disk access (MB/s)
Grid 300-100-10				
in-core	34.91	0.39	0.37	-
OOC	34.90	1.26	1.17	616

Strategy	Factor Time (sec)	Solve Fwd (sec)	Solve Bwd (sec)	Solve Disk access (MB/s)
Qimonda07				
in-core$^{(**)}$	40.4	0.9	0.9	-
OOC	191.0	186.4	207.7	13

Table 1: System based approach. $^{(**)}$ Incore time obtained on 8 processors.

In our current version of the factorisation phase only the factors are flushed to the disk and the intermediate working matrices are kept in memory. The system based approach has the advantage that on small problems (see Table 1) the observed disk bandwidth is apparently 616 MB/s. Indeed, even if the factors were written on the disk during the factorization, a significant part of them still remains in the system caches, so that the cost of accessing them during solution is the cost of a main-memory access. However, on the large problem (Qimonda07), the system cannot keep all the factors in the system caches at the end of the factorisation phase. The factor blocks then must be loaded from the disk. Furthermore, the peak speed of a memory read on the disk is 16 MB/sec, so that the minimum time to load all the factor blocks is 158 seconds. We thus see that, even in a simple context (sequential forward solution reading the disk in an identical order to that used when writing) the performance is far from the minimum. The reason is that even in this relatively simple case the system I/O mechanism is in conflict with the automatic system swapping mechanisms.

[*] ENSEEIHT-IRIT (Toulouse), Patrick.Amestoy@enseeiht.fr
[†] CERFACS (Toulouse), Iain.Duff@cerfacs.fr
[‡] INRIA Futurs-LaBRI, Abdou.Guermouche@labri.fr
[§] CERFACS (Toulouse), Mila.Slavova@cerfacs.fr

Introducing user buffers to improve performance

To solve large problems efficiently (which is the main target in designing an OOC solver), we propose to suppress the automatic use of the system (caches and prefetch) and to use smaller user buffers to explicitly control how data is prefetched from the disk. We divide the memory into two areas: a prefetching zone and an emergency one. In the prefetching zone, the whole available free space is used to load data. We prefetch each time a large enough contiguous block (1 MB in our experiments) is free in the prefetching zone. The emergency zone is used when a block factor is not prefetched or not "on the way" (part of a prefetch request). Once data is in memory, we keep it until it is used (we never reload it).

Buffers	T_factor (sec)	T_fwd (sec)	T_bwd (sec)	Nb_Req 1buffer FWD	Nb_Req EMG zone FWD	Nb_Req 1buffer BWD	Nb_Req EMG zone BWD
EMG	107.8	1149.2	1279.2	0	3 083 998	0	3 083 998
EMG+1Buffer	107.5	174.0	183.7	543	0	494	1

Table 2: Influence of the number of buffers on the uniprocessor performance on Qimonda07. EMG=emergency buffer. FWD=forward phase. BWD=backward phase. EMG buffer: 1 MB; complementary 1buffer: 10 MB.

When only the emergency buffer (EMG) is used (equivalent to a demand driven strategy) then we see that even if the FWD phase reads the factor blocks in the same order as they were written, the total number of requests to the disks incurs a very significant time overhead. Using an additional single buffer (of small size 10 MB only), our prefetching mechanism can anticipate and almost suppress the use of the emergency buffer. With this buffer and prefetch mechanism we thus show in Table 2 that we can stabilize the performance of the solve while controlling the size of the buffers being used.

Influence of scheduling on the performance and concluding remarks

In this section, we study the impact of the scheduling used during the parallel OOC solution step on its performance. A natural way to reach good performance could be to strictly follow the write sequence of the factorization step. However, the write sequences of the processors may not be compatible and can lead to deadlocks. Thus, more flexible schemes have to be used.

Strategy	Nb of Procs	T_factor (sec)	T_min (sec)	T_fwd (sec)	T_bwd (sec)	Nb_Req$^{(*)}$ 1buffer FWD	Nb_Req$^{(*)}$ emg zone FWD	Nb_Req$^{(*)}$ 1buffer BWD	Nb_Req$^{(*)}$ emg zone BWD
LIFO	1	107.5	158.4	174.0	183.7	543	0	494	1
FIFO	1	107.8	158.4	2307.9	1403.7	87	3 054 879	252	3 073 355
LIFO	2	76.0	78.8	90.0	97.4	267	0	240	4 520
LIFO	6	51.2	24.3	33.8	365.7	75	0	71	426 014
LIFO	8	42.2	20.8	24.7	212.0	61	0	32	195 990

Table 3: Influence of the scheduling of the tasks on Qimonda07. T_min corresponds to the minimum time per processor to load factors from disk. Size of EMG buffer is 1 MB; Size of the complementary buffer is 10 MB per processor; $^{(*)}$: Max per processor.

During the parallel processing of our dependency graph, a pool of ready-to-be-activated tasks is used to schedule the next task to be activated. On one processor, a LIFO (Last In First Out) order to extract tasks from the pool will lead to a contiguous access to the disk (LIFO was thus used in the previous experiments). In parallel, we cannot guarantee that the order for processing the tasks (and the factor blocks) will correspond to the order used to write them to the disks. We first see in Table 3 that on one processor the task extraction strategy has a significant impact on the performance. In parallel, even with a LIFO access, work remains to be done on the scheduling to reduce the gap between T_min and the actual time, particularly for the back substitution.
We have thus shown firstly that a "naive" system based approach is not suitable and that user buffers can be introduced to improve the performance and secondly that task scheduling is a critical issue for the parallel behaviour of the solve phase.