Dynamic scheduling for sparse direct solver on NUMA architectures

Mathieu Faverge and Pierre Ramet **

ScAlApplix Project, INRIA Bordeaux Sud-Ouest and LaBRI UMR 5800
Université Bordeaux 1, 33405 Talence cedex, France
{faverge,ramet}@labri.fr

Abstract. Over the past few years, parallel sparse direct solvers made significant progress and are now able to efficiently work on problems with several millions of equations. This paper presents some improvements on our sparse direct solver PaStiX for distributed Non-Uniform Memory Access architectures. We show results on two preliminary works: a memory allocation scheme more adapted to these architectures and a better overlap of communication by computation. We also present a dynamic scheduler that takes care of memory affinity and data locality.
Keywords. sparse direct solver, NUMA architecture, dynamic scheduling, multi-cores

1 Introduction

Over the past few years, parallel sparse direct solvers made significant progress [1, 3, 4, 6]. They are now able to solve efficiently real-life three-dimensional problems with several millions of equations. Since the last decade, most of the supercomputer architectures are based on clusters of SMP nodes. In [5], the authors proposed a hybrid MPI-thread implementation of a direct solver that is well suited for SMP nodes or modern multi-core architectures. This technique allows treating large 3D problems where the memory overhead due to communication buffers was a bottleneck for the use of direct solvers.

In the context of distributed NUMA architectures, a work has recently begun, in collaboration with the INRIA Runtime team, on studying optimization strategies, and improving the scheduling of communications, threads and I/O. Our solvers will use NewMadeleine and Marcel libraries [2, 8] in order to provide an experimental application to validate those strategies.

2 Allocation and dynamic scheduling for NUMA architecture

Our solver is based on a static scheduling [4] computed during preprocessing. To estimate the time spent in the factorisation and in the triangular solve we use

** This work is supported by the ANR grants 06-CIS-010 SOLTICE and 05-CIGC-002 NUMASIS (`http://solstice.gforge.inria.fr/` and `http://numasis.gforge.inria.fr/`)

a time model for BLAS2 and BLAS3 routines as well as for communications. A list of tasks, fully ordered, is then assigned to each available thread. This static scheduling gives very good results on most architectures. However we want to implement a dynamic scheduler at least as efficient as the static one in order to reduce some observed idle times due to approximations in our time cost models especially when communications have to be estimated. The other main objective is to preserve memory affinity and locality more particularly on nodes with a large number of cores.

It has been proved that NUMA-aware allocation can significantly improve the efficiency, hence, we change our data structure to allocate each data set needed by a thread close to its core. This avoids bottleneck on memory bus and gives faster memory access on NUMA architecture.

Secondly, it is well known that overlapping communication by computation is an important problem for most MPI applications. Some idle time still remains in our static scheduling and the efficiency could be improved by adapting dynamically the communication scheme built during the preprocessing step. Moreover, as we can see in [9], most MPI implementations do not overlap communications properly. A non-overlapped rendezvous forces a computing thread to delay the data exchange until a call to *MPI_Wait*. To avoid this problem, we ensure communication progress thanks to one or more dedicated threads.

Finally, we developed a dynamic scheduler based on the predictions from the static one. In the resulting schedule, each thread works on data close in memory. As idle times can occur, the number of threads used for computations can be increased. In that case, threads can not be bound to processors and memory access on NUMA architectures is not optimal. Therefore we plan to use the Marcel bubble scheduler [8] to group different threads and their data sets in a bubble and bind them on a part of the target architecture.

3 Numerical experiments

In this section, we describe some experiments performed on two types of architectures. For NUMA experiments, we used two dual-core opteron architectures : one with 8 processors (Hagrid) and one with 4 (Borderline), each core having 4GB of memory. Experiments on communications have been performed on a SMP cluster of IBM Power5 with 16 processors per node and connected by a "Federation" network which provides an *MPI_THREAD_MULTIPLE* implementation.

Name	Columns	NNZ_A	NNZ_L	Symmetric
AUDI	943 695	39 297 771	1.214e+09	Yes
MHD	485 597	24 233 141	1.629e+09	No

Table 1. Matrices used for our experiments

We consider two test cases (see table 1) where NNZ_A is the number of off-diagonal terms in the triangular part of the original matrix, NNZ_L is the number of off-diagonal terms in the complete factor. AUDI test case (structural mechanic problems from PARASOL collection) is a symmetric problem whereas MHD (Magneto-Hydro-Dynamic 3D problem) is an unsymmetric problem.

Threads	Allocation	Hagrid		Borderline	
		AUDI	MHD	AUDI	MHD
4	Global	**698**	**1270**	442	795
	Local	699	1300	**421**	**751**
8	Global	500	761	249	447
	Local	**363**	**662**	**217**	**408**
16	Global	386	507	-	-
	Local	**254**	**428**	-	-

Table 2. Impact of NUMA allocation (in seconds)

In table 2, we can see that binding each thread to a core and allocating its data sets close to that core (`Local` line) improve the results on NUMA architectures compared to the global allocation strategy (`Global` line). The same behaviour is also observed on our SMP cluster in a less significant way.

Proc Number	Thread Number	AUDI			MHD		
		Initial	1 Thrd	2 Thrds	Initial	1 Thrd	2 Thrds
2	1	684	**670**	672	1120	**1090**	1100
	2	388	**352**	354	594	**556**	558
	4	195	**179**	180	299	**279**	280
	8	100	**91.9**	92.4	158	**147**	**147**
	16	60.4	**56.1**	**56.1**	113	88.3	**87.4**
4	1	381	**353**	**353**	596	**559**	568
	2	191	**179**	180	304	**283**	284
	4	102	**91.2**	94.2	161	**148**	150
	8	55.5	**48.3**	54.9	98.2	**81.2**	87.3
	16	33.7	**32.2**	32.5	59.3	56.6	**56**
8	1	195	**179**	183	316	**290**	300
	2	102	**90.7**	94	187	**153**	164
	4	56.4	**47.1**	50.7	93.7	**78.8**	101
	8	31.6	**27.6**	32.4	58.4	**50**	58.7
	16	21.7	**20.4**	32.3	49.3	**41.6**	43.5

Table 3. Impact of the number of dedicated threads for communications (in seconds)

In table 3, we compare the original version and a new implementation that uses 1 or 2 specific threads to enable communication to progress. The first column is the number of MPI processes and the second column is the number of threads dedicated to computations for each process. We can see that the overlapping with threads dedicated to communications is almost better than without such dedicated threads. However, we expected that the use of two threads should have been more efficient since the network interface includes two communication cards. This can be explained by the fact that the IBM MPI implementation already uses both cards, and by its performance issues with *MPI_THREAD_MULTIPLE* described in [7].

Proc Number	Thread Number	AUDI		MHD	
		Static	Dynamic	Static	Dynamic
1	16	99.6	107	156	156
	32	**96.3**	97.8	**149**	**149**
	64	99.2	101	151	152
2	16	56.1	56.8	88.3	87
	32	55.6	**54.5**	92.5	**83.8**
	64	59	55.3	89.9	94.3
4	16	**32.2**	36.6	56.6	**54.6**
	32	33.2	35.3	70.6	68.7
	64	43.3	39.6	72	62.3

Table 4. Comparison between the static en the dynamic scheduler on SMP cluster (in seconds)

Static and dynamic schedulers are compared in table 4, both use one thread dedicated to communications and NUMA allocation is enabled. Even if our SMP cluster presents a limited NUMA effect, the dynamic scheduler allows some improvements on the execution time, especially for the unsymmetric case. Since the volume and the number of communications are more important than for the symmetric case, there is potentially more idle times to reduce.

4 Conclusion and future works

The results are encouraging since we already improved the execution time on a SMP cluster. We have to validate the approach on parallel architectures that present significant NUMA effects. We now plan to use the MARCEL bubble scheduler to improve results on NUMA architectures. Different threads and their data sets will be grouped in a bubble and bound to a part of the target architecture. The static ordering of communications has also to be adapted and, in an Out-of-Core context, new problems linked to the scheduling and the management of the computational tasks may arise (processors may be slowed down by I/O operations). Thus, we will have to design and study specific algorithms for this particular context (by extending our work on scheduling for heterogeneous platforms).

References

1. P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. 23(1):15–41, 2001.
2. O. Aumage, E. Brunet, N. Furmento, and R. Namyst. NewMadeleine: a fast communication scheduling engine for high performance networks. In *CAC 2007 held in conjunction with IPDPS 2007, Long Beach, California, USA*, March 2007.
3. Anshul Gupta. Recent progress in general sparse direct solvers. In *LNCS 2073*, pages 823–840, 2001.
4. P. Hénon, P. Ramet, and J. Roman. PaStiX: A High-Performance Parallel Direct Solver for Sparse Symmetric Definite Systems. *Parallel Computing*, 28(2):301–321, 2002.
5. P. Hénon, P. Ramet, and J. Roman. On using an hybrid MPI-Thread programming for the implementation of a parallel sparse direct solver on a network of SMP nodes. In *PPAM05, Poznan, Pologne, LNCS 3911*, pages 1050–1057, September 2005.
6. X. S. Li and J. W. Demmel. SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Trans. Mathematical Software*, 29(2):110–140, June 2003.
7. R. Thakur and W. Gropp. Test suite for evaluating performance of MPI implementations that support MPI_THREAD_MULTIPLE. In *EuroPVM/MPI, LNCS 4757*, pages 46–55, 2007.
8. S. Thibault, R. Namyst, and P.-A. Wacrenier. Building Portable Thread Schedulers for Hierarchical Multiprocessors: the BubbleSched Framework. In *EuroPar07*, Rennes, France, August 2007.
9. F. Trahay, A. Denis, O. Aumage, and R. Namyst. Improving reactivity and communication overlap in MPI using a generic I/O manager. In *EuroPVM/MPI, LNCS 4757*, pages 170–177. Springer, 2007.