Efficiently solving large sparse linear systems on a distributed and heterogeneous grid by using the multisplitting-direct method.

S. Contassot-Vivier[1], R. Couturier[1], C. Denis[2] and F. Jézéquel[2]

[1] Laboratoire d'Informatique de l'Université de Franche-Comté, CNRS FRE 2661, BP 527, 90016 Belfort CEDEX - France,
{Sylvain.Contassot-Vivier,Raphael.Couturier}@iut-bm.univ-fcomte.fr
[2] Laboratoire d'Informatique de Paris 6, CNRS UMR 7606, Université Pierre et Marie Curie - Paris 6, 4 place Jussieu, 75252 Paris CEDEX 05 - France
{Christophe.Denis,Fabienne.Jezequel}@lip6.fr

Abstract

Many scientific applications (biology, mechanics, geophysics, ...) need an increasing computational power in order to simulate phenomena close to the reality. These applications generally use existing sparse linear system libraries and tools which are efficient on sequential or parallel computers. However, heterogeneous and distributed clusters seem to be an answer to the computational power demand but bring new algorithmic problems. The purpose of the project in progress GREMLINS[3] (GRid Efficient Methods for LINear Systems) is to provide an efficient library based on direct and iterative solvers to be run on a grid. This will be done by studying the influence of pre-processing techniques such as load balancing methods. The focus will be set on studies carried out on direct methods.

Numerical algorithms have to be coarse grained in order to be efficient on a distributed grid as the communications are critically penalizing. In order to perform this, we use the so-called multisplitting method [2]. This method takes as input parameter a decomposition of the matrix into submatrices and each submatrix is taken in charge by a processor. In this method, the resolution takes an iterative form by applying on each processor a sequential method (direct or iterative) to solve its subsystem until the global result converges toward the solution of the system. In this paper, we use the routines of the `SuperLU` [3] and `MUMPS` [1] softwares.

The multisplitting method can be used either in synchronous or in asynchronous mode. Even if this method limits the amount of communications which are only performed at the end of each iteration, those frequent synchronizations slow down the performances. The asynchronous mode presents the advantage that the processors work independently and use the last data received from their neighbors. However, in order to preserve this advantage, the global detection of convergence must be modified and adapted to the asynchronous mode. For example, using 40 machines of GRID 5000 (10 in Orsay, 10 in Nancy, 10 in Toulouse, 10 in Sophia Antipolis), we obtained with `MUMPS` the results presented in Table 1 without applying the load balancing algorithm and using the synchronous and the asynchronous version. The first matrix called `cage14` is available on the Florida sparse matrix collection whereas the second one has been generated in order that the number of iterations to reach the convergence is greater. In this table, we report the size, the number of nonzero elements, the synchronous computing time, the number of iterations in the synchronous case, the asynchronous computing time and the maximum number of iterations in the asynchronous case for each matrix. In the asynchronous case, the number of iterations varies from one processor to another.

[3] The Gremlins project (http://info.iut-bm.univ-fcomte.fr/gremlins/) is supported by the French National Research Agency ANR through the "Jeunes Chercheurs" program. This work is also supported by GRID5000.

Matrix	Size	Number of nonzeros	Synchronous computing times (in s)	Number of iterations (syn. case)	Asynchronous computing times (in s)	Number of iterations (asyn. case)
cage14	1 505 785	2 7130 349	100.0	18	96.74	110
generated	300 000	5 483 999	8.60	85	4.08	310

Table 1. Computing times for the synchronous and asynchronous version with 4 sites of GRID5000

It can be noticed that although the number of iterations is greater in the asynchronous case than in the synchronous one, the execution time is smaller. The slight difference between the synchronous and asynchronous computing times for the `cage14` matrix comes from the fact that the main part of the time is consumed for the computation of the factorized form of each submatrices.

The matrix decomposition required for the multisplitting method is performed as usual with the following graph partitioning approach. First, the matrix A is modelled into a graph G. Then, a partitioning program such as METIS [5] or CHACO [4] splits the graph G into subgraphs having roughly the same number of vertices, with a few number of cut edges between subgraphs. Each subgraph corresponds to a submatrix of the matrix A having few dependencies with its neighbor submatrix. Unfortunately, although the blocks are likely to be well balanced in terms of data volumes, they may not be balanced in terms of computational volumes. We have developed a load balanced algorithm in order to better distribute the computational volume over processors or to distribute unequal but controlled computational volume to processors on a heterogeneous grid.

References

1. P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet. Hybrid scheduling for the parallel solution of linear systems. *Parallel Computing*, 32(2):136–156, 2006.
2. J. Bahi and R. Couturier. Parallelization of direct algorithms using multisplitting methods in grid environments. In *19th IEEE and ACM Int. Parallel and Distributed Processing Symposium, IPDPS 2005*, Denver, Colorado, USA, 2005. IEEE Computer Society Press.
3. J. W. Demmel, J. Gilbert, and X. S. Li. SuperLU Users Guide. Technical report, Computational Research Division, Lawrence Berkeley National Laboratory, 1997.
4. B. Hendrickson and R. Leland. The Chaco user's guide — version 2.0. Technical report, Sandia National Laboratories, Technical Report SAND94-2692, 1994.
5. G. Karypis and V. Kumar. *MeTis: Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 4.0*, September 1998.