An Hybrid Approach for the Parallelization of a Block Iterative Algorithm

Carlos Balsa[1], Ronan Guivarch[2], and Daniel Ruiz[2]

[1] CEsA–FEUP, Porto, Portugal
balsa@ipb.pt
[2] Université de Toulouse ; INP (ENSEEIHT) ; IRIT
{guivarch, ruiz}@enseeiht.fr

Abstract. The Cimmino method is a row projection method in which the original linear system is divided into subsystems. At every iteration, it computes one projection per subsystem and uses these projections to construct an approximation to the solution of the linear system.
The usual parallelization strategy in block algorithms is to distribute the different blocks on the available processors. In this paper, we follow another approach where we do not perform explicitly this block distribution to processors within the code, but let the multi-frontal sparse solver MUMPS handle the data distribution and parallelism. The data coming from the subsystems defined by the block partition in the Block Cimmino method are gathered in an unique block diagonal sparse matrix which is analysed, distributed and factorized in parallel by MUMPS. Our target is to define a methodology for parallelism based only on the functionalities provided by general sparse solver libraries and how efficient this way of doing can be.

1 Introduction

The Cimmino method is a row projection method in which the original linear system is divided into subsystems. At each iteration, it computes one projection per subsystem and uses these projections to construct an approximation to the solution of the linear system. The Block-CG method can also be used within the Block Cimmino Iteration to accelerate its convergence. Therefore, we present an implementation of a parallel distributed Block Cimmino method where the Cimmino iteration matrix is used as a preconditioner for the Block-CG.

In this work we propose to investigate a non usual methodology for parallelization of the Block Cimmino method where the direct solver package MUMPS (MUltifrontal Massively Parallel sparse direct Solver [1, 2]) is incorporated in order to schedule and perform most of the parallel tasks. This library offers to the user the facilities to call the different phases of the solver (analysis, factorization, solution) without taking care of the distribution of data and processes.

The outline of the paper is the following: the Block Cimmino Algorithm is described in section 2 and the parallelization strategy will be exposed in section 3. We finish by a presentation of some preliminary numerical results in section 4 that give us some hints for the improvements and developments still to be done.

2 Block Cimmino Algorithm

The Block Cimmino method is a generalization of the Cimmino method [3]. Basically, we partition the linear system of equations:

$$\mathbf{A}x = b, \tag{1}$$

where $\mathbf{A}$ is a $m \times n$ matrix, into l subsystems, with $l \leq m$, such that:

$$\begin{pmatrix} \mathbf{A}^1 \\ \mathbf{A}^2 \\ \vdots \\ \mathbf{A}^l \end{pmatrix} x = \begin{pmatrix} b^1 \\ b^2 \\ \vdots \\ b^l \end{pmatrix} \tag{2}$$

The block method [4] computes a set of l row projections, and a combination of these projections is used to build the next approximation to the solution of the linear system. Now, we formulate the Block Cimmino iteration as:

$$\begin{aligned} \delta^{i^{(k)}} &= \mathbf{A}^{i^+} b^i - \mathbf{P}_{\mathcal{R}(A^{i^T})} x^{(k)} \\ &= \mathbf{A}^{i^+} \left(b^i - \mathbf{A}^i x^{(k)} \right) \end{aligned} \tag{3}$$

$$x^{(k+1)} = x^{(k)} + \nu \sum_{i=1}^{l} \delta^{i^{(k)}} \tag{4}$$

In (3), the matrix $\mathbf{A}^{i^+}$ refers to the classical pseudo-inverse of $\mathbf{A}^i$ defined as: $\mathbf{A}^{i^+} = \mathbf{A}^{i^T} \left(\mathbf{A}^i \mathbf{A}^{i^T} \right)^{-1}$. However, the Block Cimmino method will converge for any other pseudo-inverse of $\mathbf{A}^i$ and in our parallel implementation we use a generalized pseudo-inverse [5], $\mathbf{A}^{i-}_{\mathbf{G}^{-1}} = \mathbf{G}^{-1} \mathbf{A}^{i^T} \left(\mathbf{A}^i \mathbf{G}^{-1} \mathbf{A}^{i^T} \right)^{-1}$, were $\mathbf{G}$ is some symmetric and positive definite matrix. The $\mathbf{P}_{\mathcal{R}(A^{i^T})}$ is an orthogonal projector onto the range of $\mathbf{A}^{i^T}$.
We use the augmented systems approach, as in [6] and [7], for solving the subsystems (3)

$$\begin{bmatrix} \mathbf{G} & \mathbf{A}^{i^T} \\ \mathbf{A}^i & 0 \end{bmatrix} \begin{bmatrix} u^i \\ v^i \end{bmatrix} = \begin{bmatrix} 0 \\ b^i - \mathbf{A}^i x \end{bmatrix} \tag{5}$$

with solution:

$$\begin{aligned} v^i &= - \left(\mathbf{A}^i \mathbf{G}^{-1} \mathbf{A}^{iT} \right)^{-1} r^i \\ u^i &= \mathbf{A}^{i-}_{\mathbf{G}^{-1}} (b^i - \mathbf{A}^i x) \\ &= \delta^i \end{aligned} \tag{6}$$

The Block Cimmino method is a linear stationary iterative method with a symmetrizable iteration matrix [8]. The symmetrized iteration matrix is symmetric

positive definite (SPD), and we can accelerate its rate of convergence with the use of Block Conjugate Gradient method (Block-CG).

The Block-CG method [9, 10] simultaneously searches for the next approximation to the system's solution in a given number of Krylov subspaces, and this number is given by the *block size* of the Block-CG method. The Block-CG method converges in a finite number of iterations in absense of roundoff errors.

3 Parallelization strategy

With block algorithms, a natural way for parallelization is to distribute the different blocks on the available processors as developed for instance in [11]. An easy distribution could be, in this case, to assign a block per processor. This natural way becomes not so easy if we want to find a distribution which aims to improve the load-balancing:

- if there is a high number of processors, we can create the same number of blocks, but the speed of convergence of such block iterative algorithms is often slowed down when the number of blocks increases;
- if the blocks are too small, the cost of communication can become prohibitive relatively to the computations.

In this paper, we follow another approach where we will get rid of this block distribution onto processors by letting the sparse direct solver package MUMPS handle the data distribution and the parallelization. The data coming from the subsystems defined by the block partition (2) in the Block Cimmino method, are gathered into a unique much larger block diagonal sparse matrix. This matrix is then given to MUMPS to be distributed, analyzed and factorized in parallel. Then, at each Block-CG iteration, MUMPS solves the system involved during the preconditioning step. Finally, we let MUMPS take care of the distribution with respect to the structure and properties of the matrix in order to have the best factorization and load balancing.

The added advantage of this way of doing, is that the sparse linear solver will handle (without any extra development for us) all the levels of parallelism available, and in particular those coming from sparsity structure and BLAS3 kernels on top of the block partitioning. This also gives us more degrees of freedom when partitioning the matrix, with the possibility to define less blocks than processors but larger ones. This may help to increase the speed of convergence of the method with still good parallelism in the end since the three levels of parallelism above are managed together.

The first step in our development was to implement the preconditioning step with the use of the MUMPS package. However, after this step, we still gather the distributed results on a master processor to perform the remaining Block-CG operations (daxpy, ddot, residual computation) in sequential. The next step to achieve our goal is to perform most of these operations in parallel by letting the data in place after distribution and factorization by MUMPS. To achieve this, we need to exploit MUMPS-embedded functionalities for data management and communications.

4 Preliminary numerical results

Some numerical experiments were carried out on the Grid'5000 platform (`https://www.grid5000.fr/mediawiki/index.php/Grid5000:Home`). Grid'5000 project aims at building a highly reconfigurable, controllable and monitorable experimental Grid platform gathering 9 sites geographically distributed in France featuring a total of 5000 processors: Bordeaux, Grenoble, Lille, Lyon, Nancy, Orsay, Rennes, Sophia-Antipolis, Toulouse. The main purpose of this platform is to serve as an experimental testbed for research in Grid Computing. This project is one initiative of the French ACI Grid incentive which provides a large part of Grid'5000 funding on behalf of the French Ministry of Research & Education.

We use the node of Toulouse, called Gridmip and especially the clusters Violette (Sun Nodes, Sun Fire V20z) and Pastel (Sun Nodes, Sun Fire Sun Fire X2200 M2).

We are currently working on the validation of the first parallel version of the code described above. The results we present are for a single matrix, AF23560. This is a matrix designed for Transient Stability Analysis of a Navier-Stokes Solver from the set of test examples AIRFOIL in the NEP Collection (`http://math.nist.gov/MatrixMarket/collections/NEP.html`).

4.1 Factorization and Solve steps

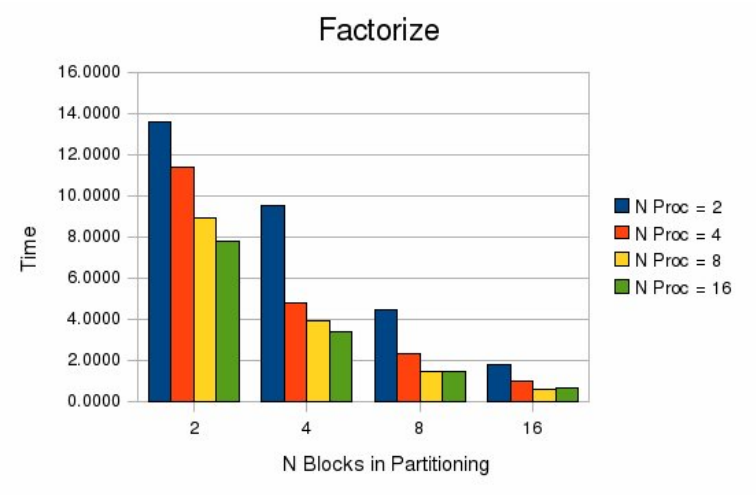

Fig. 1. Factorize Step

We can already observe the benefit from the three levels of parallelism handled together, in particular the factorization phase (Fig.1). For instance, gains

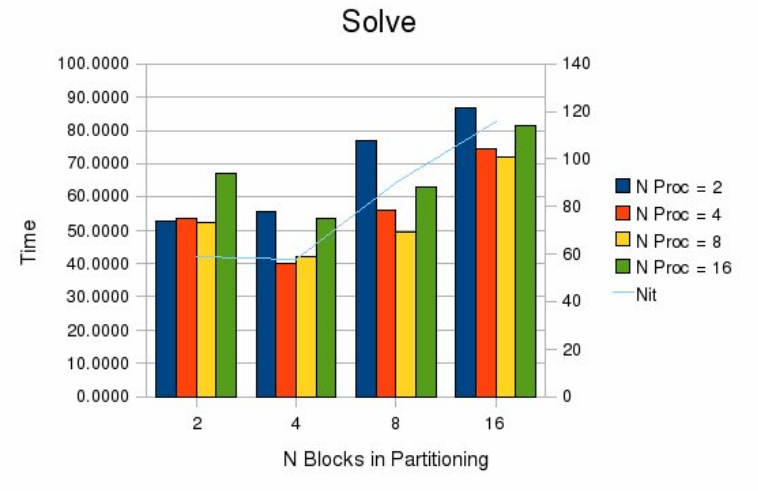

Fig. 2. Solve Step: size of Block-CG 1

in timings are obtained even with a partition in 2 blocks and with a number of processors varying from 2 to 16.

When we increase the number of blocks in the partitioning, the gains can be more marginal because the first level of parallelism, coming from the block diagonal structure in the global matrix, is already enough to feed the processors with independent computations.

The solution phase (Fig.2), however, suffers more from the sequential operations in the Block-CG iterations, which implies the gathering of the solution vector onto the master processor, and therefore a very centralized synchronization point. Still, we can observe little gains for 4 to 8 processors in the various partitionings, and the performance is not dramatically perturbed when going up to 16 processors.

The timing of the solution phase globally increases when varying the number of blocks in the partitioning, but it is simply due to the increase of the rate of convergence as we can observe in the number of iterations indicated by the curve `Nit` in Fig.2.

4.2 Influence of level 3 operations in the Block-CG

In the figures 3 and 4, we can see the effect of BLAS3 kernels when varying the number of right-hand sides in the Block-CG algorithm. This increases the granularity of the operations and enables to benefit, in the MUMPS solution phase, of larger degree of parallelism in the third level coming from the local dense operations. Still, we need to analyse in more details the reasons for the loss of performance that can be observed in the case of 16 processors.

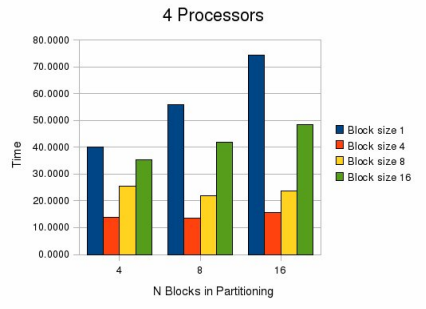

Fig. 3. Solve Step: influence of Block-CG

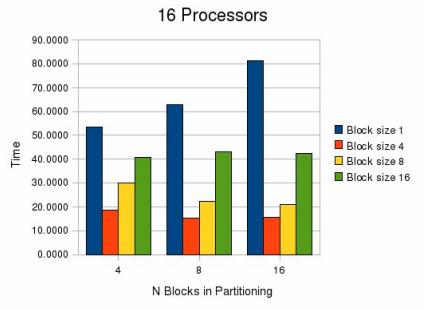

Fig. 4. Solve Step: influence of Block-CG

5 Ongoing work

We plan to perform various tests for performance, scalability, parallelism with some classical matrices from Matrix Market.

These tests will enable us to separate the parallel part, the cost of communication and the weight of the remaining sequential operations, with varying numbers of processors, matrices of different sizes and structure, etc. We will also investigate different partitioning strategies, some with few but large blocks and others with more but smaller blocks, and compare the trade-off between the three levels of parallelism managed altogether, the fill-in when factorization, and the speed of convergence that may vary from one to the other partitioning strategy.

To implement the final targeted parallel version, we will identify the basic embedded functionalities already available in the MUMPS package, such as residual computation used in iterative refinement for instance, and design new ones whenever necessary. We will also define the user interfaces to address directly these functionalities within the parallel iterative solver in order to appropriately exploit the data distribution already established and handled by MUMPS.

We plan also experiment the potential of the final solver on industrial software for the simulation of atmospheric circulation above mountain in the field of wind energy production [12].

All these different issues will be exposed in detail, and will be the core of the work we shall present in VECPAR'10.

References

1. Amestoy, P., Buttari, A., Combes, P., Guermouche, A., L'Excellent, J.Y., Pralet, S., Ucar, B.: Multifrontal massively parallel solver - user's guide of the version 4.9.4. (2009)
2. Amestoy, P., Duff, I., L'Excellent, J.Y., Koster, J.: A fully asynchronous multifrontal solver using distributed dynamic scheduling. SIAM Journal on Matrix Analysis and Applications **23** (2001) 15–41

3. Cimmino, G.: Calcolo approssimato per le soluzioni dei sistemi di equaziono lineari. In: Ricerca Sci. II. Volume 9, I. (1938) 326–333
4. Arioli, M., Duff, I.S., Noailles, J., Ruiz, D.: A block projection method for sparse matrices. SIAM Journal on Scientific and Statistical Computing (1992) 47–70
5. Campbell, S.L., Meyer, J.C.D.: Generalized inverses of linear transformations. Pitman, London (1979)
6. Bartels, R.H., Golub, G.H., Saunders, M.A.: Numerical techniques in mathematical programming. In J.B. Rosen, O. L. Mangasarian, K.R., ed.: Nonlinear Programming. Academic Press, New York (1970)
7. Hachtel, G.D.: Extended applications of the sparse tableau approach - finite elements and least squares. In Spillers, W.R., ed.: Basic question of design theory. North Holland, Amsterdam (1974)
8. Hageman, L.A., Young, D.M.: Applied Iterative Methods. Academic Press, London (1981)
9. O'Leary, D.P.: The block conjugate gradient algorithm and related methods. Linear Algebra Appl. **29** (1980) 293–322
10. Arioli, M., Ruiz, D.: Block conjugate gradient with subspace iteration for solving linear systems. In: Second IMACS International Symposium on Iterative Methods in Linear Algebra, Blagoevgrad, Bulgaria (1995) 64–79
11. Arioli, M., Drummond, A., Duff, I.S., Ruiz, D.: A parallel scheduler for block iterative solvers in heterogeneous computing environments. In: Proceedings of the seventh SIAM conference on Parallel Processing for Scientific Computing, Philadelphia, USA, SIAM (1995) 460–465
12. Silva Lopes, A., Palma, J.M.L.M., Castro, F.A.: Simulation of the Askervein flow. Part 2: Large-eddy simulations. Boundary-Layer Meteorology **125** (2007) 85–108