Journal of Computational Science 27 (2018) 255–270

Contents lists available at ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

Sparse supernodal solver using block low-rank compression: Design, performance and analysis

Grégoire Pichon[a,*], Eric Darve[b], Mathieu Faverge[a], Pierre Ramet[a], Jean Roman[a]

[a] Inria, CNRS (LaBRI UMR 5800), Bordeaux INP, Université de Bordeaux, 33400 Talence, France
[b] Mechanical Engineering Department, Stanford University, United States

ARTICLE INFO

Article history:
Received 28 December 2017
Received in revised form 11 June 2018
Accepted 13 June 2018
Available online 18 June 2018

Keywords:
Sparse linear solver
Block low-rank compression
PaStiX sparse direct solver
Multi-threaded architectures

ABSTRACT

This paper presents two approaches using a Block Low-Rank (BLR) compression technique to reduce the memory footprint and/or the time-to-solution of the sparse supernodal solver PaStiX. This flat, non-hierarchical, compression method allows to take advantage of the low-rank property of the blocks appearing during the factorization of sparse linear systems, which come from the discretization of partial differential equations. The proposed solver can be used either as a direct solver at a lower precision or as a very robust preconditioner. The first approach, called *Minimal Memory*, illustrates the maximum memory gain that can be obtained with the BLR compression method, while the second approach, called *Just-In-Time*, mainly focuses on reducing the computational complexity and thus the time-to-solution. Singular Value Decomposition (SVD) and Rank-Revealing QR (RRQR), as compression kernels, are both compared in terms of factorization time, memory consumption, as well as numerical properties. Experiments on a shared memory node with 24 threads and 128 GB of memory are performed to evaluate the potential of both strategies. On a set of matrices from real-life problems, we demonstrate a memory footprint reduction of up to 4 times using the *Minimal Memory* strategy and a computational time speedup of up to 3.5 times with the *Just-In-Time* strategy. Then, we study the impact of configuration parameters of the BLR solver that allowed us to solve a 3D laplacian of 36 million unknowns a single node, while the full-rank solver stopped at 8 million due to memory limitation.

1. Introduction

Many scientific applications such as electromagnetism, geophysics or computational fluid dynamics use numerical models that require to solve linear systems of the form $Ax = b$, where the matrix A is sparse and large. In order to solve these problems, a classic approach is to use a sparse direct solver which factorizes the matrix into a product of triangular matrices before solving triangular systems.

Yet, there are still limitations to solve larger and larger systems in a black-box approach without any knowledge of the geometry of the underlying partial differential equation. Memory requirements and time-to-solution limit the use of direct methods for very large matrices. On the other hand, for iterative solvers, general black-box preconditioners that can ensure fast convergence for a wide range of problems are still missing.

In the context of sparse direct solvers, some recent works have investigated the low-rank representations of dense blocks appearing during the sparse matrix factorization, by compressing blocks through many possible compression formats such as Block Low-Rank (BLR), $\mathcal{H}$, $\mathcal{H}^2$, HSS, HODLR... These different approaches reduce the memory requirement and/or the time-to-solution of the solvers. Depending on the compression strategy, these solvers require knowledge of the underlying geometry to tackle the problem or can do it in a purely algebraic fashion.

Hackbusch [1] introduced the $\mathcal{H}$-LU factorization for dense matrices. It compresses the matrix into a hierarchical matrix format before applying low-rank operations instead of classic dense operations. The dense solver was extended to consider sparse matrices by using a nested dissection ordering to exhibit the hierarchical structure, as summarized in [2].

In [3], $\mathcal{H}$-LU factorization is used in an algebraic context. Performance, as well as a comparison of $\mathcal{H}$-LU with some sparse direct solvers is presented in [4]. Kriemann [5] and Lizé [6] implemented this algorithm using Directed Acyclic Graphs.

The Hierarchically Off-Diagonal Low-Rank (HODLR) compression technique was used in [7] to accelerate the elimination of large dense fronts appearing in the multifrontal method. It was fully integrated in a sparse solver in [8], and uses Boundary Distance Low-Rank (BDLR) to allow both time and memory savings, by avoiding the formation of dense fronts.

* Corresponding author.

https://doi.org/10.1016/j.jocs.2018.06.007
1877-7503/© 2018 Elsevier B.V. All rights reserved.

A supernodal solver using a compression technique similar to HODLR was presented in [9]. The proposed approach allows memory savings and can be faster than standard preconditioning techniques. However, it is slower than the direct approach in the benchmarks and requires an estimation of the rank to use randomized techniques and to accelerate the solver.

The use of Hierarchically Semi-Separable (HSS) matrices in sparse direct solvers has been investigated in several solvers. In [10], Xia et al. presented a solver for 2D geometric problems, where all operations are computed algebraically. In [11], a geometric solver was developed, but contribution blocks are not compressed, making memory savings limited. [12,13] proposed an algebraic code that uses randomized sampling to manage low-rank blocks and to allow memory savings.

H^2 arithmetic [14] has been used in several sparse solvers. In [15], a fast sparse H^2 solver, called LoRaSp, based on extended sparsification was introduced. In [16], a variant of LoRaSp, aiming at improving the quality of the solver when used as a preconditioner, was presented, as well as a numerical analysis of the convergence with H^2 preconditioning. In particular, this variant was shown to lead to a bounded number of iterations irrespective of problem size and condition number (under certain assumptions). In [17] a fast sparse solver was introduced based on interpolative decomposition and skeletonization. It was optimized for meshes that are perturbations of a structured grid. In [18], an H^2 sparse algorithm was described. It is similar in many respects to [15], and extends the work of [17]. All these solvers have a guaranteed linear complexity, for a given error tolerance, and assuming a bounded rank for all well-separated pairs of clusters (the admissibility criterion in Hackbusch et al.'s terminology).

Block Low-Rank compression has been investigated for dense matrices [19,20], and for sparse linear systems when using a multifrontal method [21,22]. Considering that these approaches are similar to the current study, a detailed comparison will be described in Section 6. One of the differences of our approach with [22] is the supernodal context that leads to different low-rank operations, and possibly increases the memory savings. The main contribution of our work is the use of low-rank assemblies to avoid forming dense updates and to maximize memory savings.

The first objective of this work is to combine a generic sparse direct solver with recent work on matrix compression to solve larger problems, overcoming the memory limitations and accelerating the time-to-solution. The second objective is to keep the black-box algebraic approach of sparse direct solvers, by relying on methods that are independent of the underlying problem geometry. In this paper, we consider the multi-threaded sparse direct solver PaStiX [23] and we introduce a BLR compression strategy to reduce its memory and computational cost. We developed two strategies: *Minimal Memory*, which focuses on reducing the memory consumption, and *Just-In-Time* which focuses on reducing the time-to-solution (factorization and solve steps).

During the factorization, the first strategy compresses the sparse matrix before factorizing it, *i.e.* compresses A factors, and exploits dedicated low-rank numerical operations to keep the memory cost of the factorized matrix as low as possible. The second strategy compresses the information as late as possible, *i.e.* compresses L factors, to avoid the cost of low-rank update operations. The resulting solver can be used either as a direct solver for low accuracy solutions or as a high-accuracy preconditioner for iterative methods, requiring only a few iterations to reach the machine precision. The main contribution of this work is the introduction of low-rank compression in a supernodal solver with a purely algebraic method. Indeed, contrary to [9] which uses rank estimations (*i.e.* a non-algebraic criteria), our solver computes suitable ranks to maintain a prescribed accuracy.

A preliminary version of this work appeared in [24]. In this paper, we introduce new orthogonalization methods for the RRQR recompression kernels which leads to a better limitation of the rank growth during the factorization. We present a detailed analysis of the solver with a parallelism study and a comparison of efficiency of the low-rank kernels with respect to the original full-rank kernels. We also evaluate the impact of several parameters on the memory consumption and time to solution such as the blocking sizes, the maximum rank accepted for low-rank forms and the different orthogonalization methods. In Section 2, we go over basic aspects of sparse supernodal direct solvers. The two strategies, introduced in PaStiX, are then presented in Section 3, before detailing low-rank kernels in Section 4. Section 5 compares the two BLR strategies with the original approach, that uses only dense blocks, in terms of memory consumption, time-to-solution and numerical behaviour. We also investigate the efficiency of low-rank kernels, as well as the impact of the BLR solver parameters. Section 6 surveys in more detail related works on BLR for dense and/or sparse direct solvers, highlighting the differences with our approach, before discussing how to extend this work to a hierarchical format (H, HSS, HODLR...).

2. Background on sparse linear algebra

The common approach used by sparse direct solvers is composed of four main steps: (1) ordering of the unknowns, (2) computation of a symbolic block structure, (3) numerical block factorization, and (4) triangular system solves. In the rest of the paper, we consider that all problems have a symmetric pattern given by the pattern of $A+A^t$.

The purpose of the first step is to minimize the fill-in — zero becoming non-zero — that occurs during the numerical factorization to reduce the number of operations as well as the memory requirements to solve the problem. In order to both reduce fill-in and exhibit parallelism, the nested dissection [25] algorithm is widely used through libraries such as Metis [26] or Scotch [27]. Each set of vertices corresponding to a separator constructed during the nested dissection is called a supernode.

From the resulting supernodal partition, the second step predicts the symbolic block structure of the final factorized matrix (L) and the block elimination tree. This block structure is composed of one block of columns (column block) for each supernode of the partition, with a dense diagonal block and several dense off-diagonal blocks, as presented in Fig. 1 for a 3D Laplacian.

The goal is to exhibit large block structures to leverage efficient Level 3 BLAS kernels during the numerical factorization. However, one may notice (cf. Fig. 1) that the symbolic structure obtained with a general partitioning tool might be composed of many small off-diagonal blocks contributing to larger blocks. These off-diagonal blocks might be grouped together by adding zero to the structure if the BLAS efficiency gain is worthwhile and if the memory overhead induced by the fill-in is limited. Alternatively, it is also possible to reorder supernode unknowns to group off-diagonal blocks together without additional fill-in. A traveling salesman strategy is implemented in PaStiX [28] and divides by more than two the number of these off-diagonal blocks. Other approaches [12,21] perform a k-way ordering of supernodes, starting from a reconnected graph of a separator, to order consecutively vertices belonging to a same local part of the separator's graph. Such reordering technique also allows to reduce ranks of the low-rank blocks as shown in [21]. To introduce more parallelism and data locality, the final structure can then be split into tiles as it is now commonly done in dense linear algebra libraries to fit the computational units granularity. These first two steps of direct solvers are preprocessing stages independent from the numerical values. Note that these steps can be computed once