Parallel Computing 000 (2017) 1–16

Contents lists available at ScienceDirect

Parallel Computing

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

A distributed-memory hierarchical solver for general sparse linear systems

Chao Chen[a,*], Hadi Pouransari[b], Sivasankaran Rajamanickam[c], Erik G. Boman[c], Eric Darve[a,b]

[a] Institute for Computational and Mathematical Engineering, Stanford University, Stanford, USA
[b] Department of Mechanical Engineering, Stanford University, Stanford, USA
[c] Center for Computing Research, Sandia National Laboratories, Albuquerque, USA

ARTICLE INFO

Article history:
Received 22 December 2016
Revised 7 November 2017
Accepted 19 December 2017
Available online xxx

Keywords:
Parallel linear solver
Sparse matrix
Hierarchical matrix

ABSTRACT

We present a parallel hierarchical solver for general sparse linear systems on distributed-memory machines. For large-scale problems, this fully algebraic algorithm is faster and more memory-efficient than sparse direct solvers because it exploits the low-rank structure of fill-in blocks. Depending on the accuracy of low-rank approximations, the hierarchical solver can be used either as a direct solver or as a preconditioner. The parallel algorithm is based on data decomposition and requires only local communication for updating boundary data on every processor. Moreover, the computation-to-communication ratio of the parallel algorithm is approximately the volume-to-surface-area ratio of the subdomain owned by every processor. We present various numerical results to demonstrate the versatility and scalability of the parallel algorithm.

1. Introduction

Solving large sparse linear systems is an important building block – but often a computational bottleneck – in many engineering applications. Large sparse linear systems arise, for example, from local discretization of elliptic partial differential equations. Solving a general linear system that has N non-zeros with $O(N)$ computer memory and CPU time is challenging, especially when the underlying physical problem is three-dimensional. Existing methods fall into three categories as follows.

The first category of methods are sparse direct solvers [1] based on Gaussian elimination. These methods organize computation efficiently and leverage fill-reducing ordering schemes. For example, the nested dissection multifrontal algorithm [2,3] performs elimination according to a specific hierarchical structure of the unknowns. Because of their robustness and efficiency, several state-of-the-art sparse direct solvers have been implemented into software packages, which target sequential [4–6], shared-memory [7,8] and distributed-memory computers [9,10]. However, sparse direct solvers generally require $O(N^2)$ work and $O(N^{\frac{4}{3}})$ computer memory for a three-dimensional problem of size N. This quadratic factorization cost and large memory footprint seriously limit the application of sparse direct solvers to truly large-scale problems.

The second category consists of iterative solvers. These solvers require only $O(N)$ computer memory to store the linear system and are thus more memory-efficient than sparse direct solvers. They can also achieve the optimal time-complexity if

* Corresponding author.
E-mail addresses: cchen10@stanford.edu (C. Chen), hadip@stanford.edu (H. Pouransari), srajama@sandia.gov (S. Rajamanickam), egboman@sandia.gov (E.G. Boman), darve@stanford.edu (E. Darve).

https://doi.org/10.1016/j.parco.2017.12.004
0167-8191/

the number of iterations is small. For example, the multigrid method [11] is typically the fastest solver for many discretized elliptic PDEs. However, iterative solvers have three disadvantages. First, the convergence of iterative solvers is not guaranteed for general linear systems. For example, the multigrid method may fail to converge for indefinite linear systems and linear systems coming from coupled PDEs. Second, the number of iterations may grow rapidly as the condition number of a linear system increases. Third, the setup phase of some iterative solvers relies on sparse matrix-sparse matrix multiplication, as in the algebraic multigrid method, which is complex to scale [12,13].

The third category of methods developed for solving sparse linear systems are hierarchical solvers [14–19] and their parallel counterparts [20–23]. The general idea behind these hierarchical solvers is exploiting the low-rank structure of dense matrix blocks that arise during the elimination process to reduce storage and computational cost. As a result of utilizing the data sparsity, hierarchical solvers, compared with sparse direct solvers, typically have reduced memory footprint and smaller computational complexity. However, some of the existing hierarchical solvers still cannot attain quasilinear complexity for solving three-dimensional problems, and others may be either too complicated to be implemented efficiently or may be restricted to only structured problems.

In this paper, we introduce a new parallel hierarchical solver for solving general sparse linear systems. Our parallel algorithm is based on the sequential algorithm in [19] called LoRaSp, which computes an approximate factorization with sparse block-triangular factors. In particular, our method eliminates the unknowns cluster by cluster and compresses fill-in blocks in a hierarchical fashion. The singular values of these fill-in blocks are often found to decrease geometrically, and general algebraic techniques, such as the SVD, can be used to compute corresponding low-rank approximations. Since the dropping/truncation rule in our method is based on the decay of singular values, it is expected to be more efficient than other level-based or threshold-based rules, which are typically used in the incomplete LU (ILU) factorization [24]. In our method, the bases computed in low-rank approximations serve the same role as restriction and prolongation operators do in the algebraic multigrid method (AMG) [25,26]. While the construction of restriction and prolongation operators in AMG may require tuning and manual adjustments for a specific linear system, the low-rank bases in our method are computed in a systematic fashion, regardless of the underlying PDE or physical problem.

There are two differences between our method and other hierarchical solvers. First, while most of hierarchical solvers are developed under either the $\mathcal{H}$- [27,28] or the hierarchically semiseparable (HSS) [29,30] matrix frameworks, our method is built upon the $\mathcal{H}^2$-matrix theory [31,32], which provides a more efficient hierarchical low-rank structure. Under some mild conditions, the computational cost and the memory footprint of our method scale linearly with respect to the problem size, and we observed quasilinear complexity in practice for solving various types of problems. Second, unlike other hierarchical solvers, which are typically combined with the multifrontal algorithm, our method relies on domain partitioning, which naturally leads to a data decomposition scheme in the parallel algorithm. Put another way, these two differences allow our parallel algorithm to have the following three features:

1. Only local communication is required for every processor.
2. The computation-to-communication ratio is approximately the volume-to-surface-area ratio of the subdomain owned by a processor.
3. The bulk of computation is from using sequential dense linear algebra, which has the potential to be significantly accelerated on modern many-core architectures.

To summarize, this paper presents a parallel hierarchical solver for general sparse linear systems, and especially, our work makes the following three major contributions:

1. New derivation of the LoRaSp algorithm, which reveals the structure of calculation and data dependency in the original algorithm;
2. Development of a bulk-synchronous parallel algorithm and a task-based asynchronous parallel algorithm with the optimal scheduling strategy;
3. Development of a coloring scheme to extract maximum concurrency in the execution, and discussion on optimizing load-balancing in the presence of coloring constraints.

The remainder of this paper is organized as follows. Section 2 presents the sequential algorithm, a new derivation of LoRaSp. Section 3 presents the parallel algorithm, focusing on techniques to keep communication local and to maximize concurrency. Section 4 analyzes the computation and communication cost of the parallel algorithm. Section 5 presents numerical results to demonstrate the versatility and parallel scalability of our parallel hierarchical solver.

2. Sequential algorithm

This section presents our new derivation of the LoRaSp algorithm. Although the algorithm works for a general sparse linear system $Ax = b$, we focus on symmetric positive definite (SPD) systems for ease of presentation. From a high-level perspective, the algorithm computes an approximate factorization of an SPD matrix A with the following steps.

First, a partitioning of the rows/columns of A is computed algebraically. Suppose Π is the set of row/column indices (DOFs). A clustering $\Pi = \cup_i \pi_i$, where π_i is a cluster of DOFs, can be computed with a graph partitioner, such as METIS/ParMETIS [33], Scotch [34] and Zoltan [35]. Second, some portions of DOFs in every cluster are eliminated as all clusters in $\Pi = \cup_i \pi_i$ are looped over. Specifically, the fill-in blocks associated with a cluster π_s are compressed, and π_s is split

Please cite this article as: C. Chen et al., A distributed-memory hierarchical solver for general sparse linear systems, Parallel Computing (2017), https://doi.org/10.1016/j.parco.2017.12.004