Contents lists available at ScienceDirect

Comput. Methods Appl. Mech. Engrg.

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

Sparse direct factorizations through unassembled hyper-matrices ☆

Paolo Bientinesi [a], Victor Eijkhout [b,*], Kyungjoo Kim [c], Jason Kurtz [d], Robert van de Geijn [e]

[a] Aachen Institute for Computational Engineering Science, RWTH Aachen, Germany
[b] Texas Advanced Computer Center, The University of Texas at Austin, United States
[c] Aerospace Engineering and Engineering Mechanics, The University of Texas at Austin, United States
[d] Applied Research Laboratories, The University of Texas at Austin, United States
[e] Computer Science Department, The University of Texas at Austin, United States

ARTICLE INFO

Article history:
Received 15 November 2007
Received in revised form 24 June 2009
Accepted 10 July 2009
Available online 12 August 2009

Keywords:
Factorizations
Gaussian elimination
Sparse matrices
hp-Adaptive finite elements

ABSTRACT

We present a novel strategy for sparse direct factorizations that is geared towards the matrices that arise from *hp*-adaptive Finite Element Methods. In that context, a sequence of linear systems derived by successive local refinement of the problem domain needs to be solved. Thus, there is an opportunity for a factorization strategy that proceeds by updating (and possibly downdating) the factorization. Our scheme consists of storing the matrix as unassembled element matrices, hierarchically ordered to mirror the refinement history of the domain. The factorization of such an 'unassembled hyper-matrix' proceeds in terms of element matrices, only assembling nodes when they need to be eliminated. The main benefits are efficiency from the fact that only updates to the factorization are made, high scalar efficiency since the factorization process uses dense matrices throughout, and a workflow that integrates naturally with the application.

1. Introduction

Many scientific applications spend a large amount of time in the solution of linear systems, often performed by sparse direct solvers. We argue that traditional matrix storage schemes, whether dense or sparse, are a bottleneck, limiting the potential efficiency of the solvers. We propose a new data structure, the unassembled hyper-matrix (UHM). This data structure preserves useful information that can be provided by the application, and that can make the solver, as well as various other operations on the matrix, more efficient. In particular, we will use this storage format to implement an efficient sparse direct solver for *hp*-adaptive[1] finite element method (FEM) problems.

The improvement in efficiency will come from rethinking the conventional approach where sparse direct solvers are used as a black-box, where a linear system is passed as input and a solution is returned as output. Much progress has been made on making such a black-box procedure as efficient as possible. However, traditional solvers are intrinsically handicapped by ignoring domain information. Furthermore, what is not exploited by such solvers is the fact that once a solution for a given discretized problem has been computed, modifications to this existing discretization are made. This means that what should be the real measure of efficiency is how fast a solution of a somewhat modified (refined) problem can be computed given a factorization of the current problem (discretization). We argue that this formulation of the problem leads to dramatically different data structures and factorization approaches, on which the existing literature on sparse direct solvers has little bearing.

In this introduction we sketch the demands on a matrix storage scheme in a FEM application, and show how traditional linear algebra software insufficiently addresses these demands. In the rest of this paper we will then show how the UHM scheme overcomes these limitations. For the time being, we limit ourselves to symmetric positive-definite (SPD) problems.

1.1. The workflow of advanced FEM solvers

Adaptive discretization techniques are recognized to be the key to the efficient and accurate solution of complicated FEM problems, for instance, problems with singularities around re-entrant corners. (We will give a brief overview of adaptive and in particular *hp*-adaptive FEM in Section 2.)

☆ This work was supported by NSF Grant #DMS-0625917. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF).

* Corresponding author. Tel.: +1 512 471 5809; fax: +1 512 475 9445.
E-mail address: eijkhout@tacc.utexas.edu (V. Eijkhout).

[1] The designation '*hp*' refers to the simultaneous refinement of the space discretization *h*, and the polynomial degree *p*.

A typical *hp*-adaptive FE computation proceeds as follows.

1. An initial discretization is generated to represent the given geometry and material data.
2. The global stiffness matrix and load vector are computed, stored either in an assembled sparse format or as unassembled element contributions.
3. The sparse linear system is solved via a standard solver package. Here the choice of package depends on whether the solution is computed via a direct or iterative solver. For sparse direct solution, widely used packages include MUMPS [1,30], NASTRAN (various commercial versions), SuperLU [12,13,26], and UMFPACK [8,37]. For iterative solution current favorites include PETSc [2,3] and Trilinos [20].
4. Based on *a posteriori* error estimates, it is determined whether to break or merge elements (*h*-refinement or unrefinement) and/or whether to increase (or decrease) the order of approximation *p*. These decisions are made locally, i.e., on an element-by-element basis, though refinement of one element may induce surrounding elements to be refined too, in some circumstances.
5. Steps 2–4 are repeated until a stopping criterion is met.

1.2. Shortcomings of the existing approach

There are two essential limitations with the traditional approach to FEM solvers outlined above.

One obvious problem is in Step 3, where the solver has no way of knowing whether it was invoked before, and what the relation is between its input data in successive invocations. Since successive linear systems are clearly related, considerable opportunity for efficiency is left unexplored.

There is a further problem in that, by formulating the linear system as a matrix equation, much information about the application is lost. This information is then laboriously, and imperfectly, reconstructed by the graph partitioners used in sparse solver packages.

These shortcomings of the matrix-based interface are not purely academic. In Section 2.3 we will show anecdotal evidence that fairly simple manual preprocessing of the linear systems can significantly improve the efficiency of a standard direct solver. Clearly, certain knowledge of the linear system that is available to a human can only imperfectly be discovered by a black-box solver. Our improved data structure and solver preserve and exploit such knowledge.

1.2.1. Inflexibility in an application context

Current linear algebra software has little provision for the reality that often a sequence of related linear systems is to be solved. Even a slight change to the matrix forces a solver package to recompute the factorization from scratch, with no information preserved.

However, in the setting of adaptive FEM solvers, the next linear system is often derived from the previous one by refining part of the physical domain, either in space, or in the order of the FEM basis functions. Traditional matrix storage is not flexible enough to accommodate insertion of matrix rows easily. Instead, a whole new matrix needs to be allocated, with the old data copied over or even recomputed, at considerable overhead. Furthermore, solver packages cannot preserve parts of a factorization that are not affected by such a refinement. Our UHM storage scheme remedies both shortcomings.

While it can be argued that the use of a high-quality graph partitioner favors the current approach to successive substructuring for a single solution, storing the stiffness matrix as an UHM conformal to the hierarchy in the domain has the potential for greatly reducing the cost of subsequent solves with refined data. There is some memory overhead associated with element-by-element (EBE) codes: anecdotal evidence suggests 30% for matrix storage in a 2D case with low polynomial degree [6,5], and possibly more with higher degrees, see Table 1 in [32]. However, this is outweighed by advantages in performance and flexibility. Also, the overhead from EBE storage for the factorization is considerably less.

1.2.2. Loss of application information

The representation of a matrix as a two-dimensional array of numbers, whether stored densely or using a sparse storage format, represents a bottleneck between the application and the solver library. Relevant application knowledge is lost, such as geometry and other properties of the domain, various facts about the nature of the mesh, and any history of refinement that led to the current system of equations. Much of the development of sparse direct solvers goes into reconstructing, algebraically, this information.

1.3. Relation to existing factorizations

Our factorization scheme contains some novel elements, foremost the fact that we retain the refinement history of the Finite Element (FE) grid. Of course, there are various connections to the existing literature. In this section we highlight a few. (For a recent overview of the field of sparse direct factorizations, see the book by Davis [9].)

1.3.1. Substructuring

Techniques of bisection and recursive bisection have long been a successful strategy, although not the only one, for deriving direct solvers. Initial research on solvers on a regular domain showed considerable savings in storage for two-dimensional problems [16,17]. These results have been extended to arbitrary finite element meshes [27,28], including a proof that in the three-dimensional case no order improvement exists as in the 2D case: in 2D, the naïve space bound of $O(N^{3/2})$ can be improved to $O(N \log N)$; in 3D, no reduction of the naïve $O(N^{5/3})$ bound is known. More recently, spectral bisection techniques have been explored as a way of deriving multiple partitioning of a set of variables [15,18,19,22,34]. Another direction in graph partitioning is that of partitioning methods based on space-filling curves [31,33]. These methods have been used primarily for partitioning elements in work related to iterative solvers. Again, such techniques are based on algebraic properties of the matrix graph, and can only imperfectly reconstruct any division that is natural to the problem.

1.3.2. Supernodes

With the realization that Level 3 Basic Linear Algebra Subprograms (BLAS [24,7]) operations are the path to high performance in linear algebra codes (see for instance [14,4]), researchers of sparse direct solvers have devoted considerable effort to finding ‘supernodes’: blocks of rows or columns that have similar sparsity patterns, and can thus be tackled with dense block algorithms [36,25] when combined. However, this block structure derives from the elements in a Finite Element mesh, so we conclude that the linear algebra software aims at reconstructing information that was present in the application and was lost in the traditional solver interface.

It is clear that a matrix representation that preserves information about the operator and the discretization has a potential advantage over traditional storage formats.

1.3.3. Hierarchical methods

The idea of using a tree structure in the factorization of a matrix has occurred to several authors and in several contexts. However,