Computers and Structures 141 (2014) 19–25

Contents lists available at ScienceDirect

Computers and Structures

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

Evaluation of massively parallel linear sparse solvers on unstructured finite element meshes

Seid Koric [a,b,*], Qiyue Lu [a], Erman Guleryuz [a]

[a] National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign, 1205 W. Clark St., Urbana, IL 61801, USA
[b] Department of Mechanical Science and Engineering, University of Illinois at Urbana-Champaign, 1206 W. Green St., Urbana, IL 61801, USA

ARTICLE INFO

Article history:
Received 26 November 2013
Accepted 12 May 2014
Available online 11 June 2014

Keywords:
Sparse linear solvers
Direct and iterative methods
High performance computing
Parallel Speedup
Finite element method
Unstructured mesh

ABSTRACT

The performance of massively parallel direct and iterative methods for solving large sparse systems of linear equations arising in finite element method on unstructured (free) meshes in solid mechanics is evaluated on a latest high performance computing system. We present a comprehensive comparison of a representative group of direct and iterative sparse solvers. Solution time, parallel scalability, and robustness are evaluated on test cases with up to 40 million degrees of freedoms and 3.3 billion nonzeros. The results show that direct solution methods, such as multifrontal with hybrid parallel implementation, as well as new hybrid adaptive block factorized preconditioning iterative methods can take a full advantage of a modern high performance computing system and provide superior solution time and parallel scalability performance.

1. Introduction

Solving linear system of equations:

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{1}$$

lies at the heart of many problems in computational science and engineering and is responsible for 70–80% of the total computational time. In many cases, particularly when discretizing continuous solid mechanics problems with implicit finite element method, the associated matrix $\boldsymbol{A}$ is sparse, symmetric and positive definite (SPD). Linear problems are solved with a single solution of Eq. (1). Within each quasi-static time step of nonlinear problems, however, a system of nonlinear equations is linearized and solved with a Newton–Raphson (NR) iteration scheme [1,2], which requires several linear solver solutions of global equilibrium iterations (subscript i) as follows:

$$\left[K_{i-1}^{t+\Delta t}\right]\left\{\Delta u_{i-1}^{t+\Delta t}\right\} = \left\{R_{i-1}^{t+\Delta t}\right\} \tag{2}$$

where $\left\{\Delta u_{i-1}^{t+\Delta t}\right\}$ is the incremental change to the solution vector (displacements in mechanical problems), and $\left\{R_{i-1}^{t+\Delta t}\right\}$ is the residual error vector. A linear solver is used to solve Eq. (2) for $\left\{\Delta u_{i-1}^{t+\Delta t}\right\}$, which is used to update the solution vector in Eq. (3), until convergence is achieved everywhere at time $t + \Delta t$ (i.e., when the update vector is sufficiently small).

$$\left\{u_i^{t+\Delta t}\right\} = \left\{u_{i-1}^{t+\Delta t}\right\} + \left\{\Delta u_{i-1}^{t+\Delta t}\right\} \tag{3}$$

The tangent stiffness matrix $[K^{t+\Delta t}]$ is defined in Eq. (5) from the consistent tangent operator, also known as the material Jacobian, $[J]$, which is defined in Eq. (4) for mechanical problems, taking $\Delta\hat{\varepsilon}^{t+\Delta t}$ as a guessed mechanical strain increment, based on the current best displacement increment.

$$\underline{\underline{\mathbf{J}}} = \frac{\partial \Delta \boldsymbol{\sigma}^{t+\Delta t}}{\partial \Delta \hat{\boldsymbol{\varepsilon}}^{t+\Delta t}} \tag{4}$$

$$[K^{t+\Delta t}] = \int_V [B]^t [J][B] \mathrm{dV} \tag{5}$$

where $[B] = \partial[N]/\partial\mathbf{x}$ contains the spatial derivatives of the element shape functions $[N]$.

For more than three decades, there has been considerable interest in the development of numerical algorithms for the solution of large sparse linear systems of equations and their efficient parallel implementation on high performance computing systems. The algorithms may be grouped into two broad categories: direct methods and iterative methods.

Iterative method algorithms repeatedly apply a sequence of operations at each step attempting to improve upon its current approximation to a solution. Krylov subspace methods are an

* Corresponding author at: National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign, 1205 W. Clark St., MC-257, Urbana, IL 61801, USA. Tel.: +1 (217) 265 8410.
E-mail addresses: koric@illinois.edu (S. Koric), qiyuelu1@ncsa.illinois.edu (Q. Lu), guleryuz@illinois.edu (E. Guleryuz).

http://dx.doi.org/10.1016/j.compstruc.2014.05.009
0045-7949/

important class of iterative methods. This class includes the Conjugate Gradient (CG) method [3,4] and it variants, which is robust for SPD matrices. In solving the large systems in finite element method, combining a Krylov subspace method such as CG with a preconditioner is essential to accelerate convergence rate and avoid divergence of solution especially for ill-conditioned linear systems.

The most widely used direct methods [5], are variants of Gaussian elimination and involve the explicit factorization of the system matrix $\boldsymbol{A}$ (or, more usually, a permutation of $\boldsymbol{A}$) into a product of lower and upper triangular matrices $\boldsymbol{L}$ and $\boldsymbol{U}$. In the symmetric case, $\boldsymbol{U} = \boldsymbol{D}\boldsymbol{L}^T$, where $\boldsymbol{D}$ is a block diagonal matrix with 1×1 and 2×2 matrix blocks. Ordering phase reorders the rows and columns such that the factors have minimal fill-in. Symbolic factorization phases analyze the matrix structure to determine an optimal pivoting sequence and strategy for optimal factorization. Forward elimination (factorization) followed by backward substitution completes the solution process for each given right-hand side $\boldsymbol{b}$. The main advantages of direct methods are their generality and robustness.

For some tough (ill-conditioned) linear systems that arise in a number of application areas, direct methods are currently the only feasible methods. For other problems, finding and computing a good preconditioner for use with an iterative method can be computationally more expensive than using a direct method. In case of nonlinear problems, the matrix structure does not change for the linear solvers within each NR nonlinear iteration in Eq. (2). Ordering and symbolic factorization phases are performed only for the first NR iteration for each step. For every subsequent nonlinear iteration, only the factorization and backward-solve need to be called with direct solvers.

A significant weakness of direct methods, however, is that the matrix factors are often significantly denser than the original matrix, and for large problems such as those that arise from discretization of three dimensional partial differential equations, insufficient memory for both forming and then storing the factors can prevent the use of direct methods. The limitation on CPU and memory requirements had made the use of direct solvers uneconomical in the past, resulting in broad use of iterative solvers. The recent rise of terascale and especially petascale computational resources, however, as well as the development of multifrontal [6] and supernodal techniques [7], has greatly increased the efficiency and practicality of using direct solvers for large sparse systems.

2. Solvers, test cases, and computing platform

Key features of the solver libraries used in study are listed in Table 1. Two iterative libraries were used in this work: PETSc [8] and Hypre [9], Both provide parallel Message Passing Interface (MPI) routines for solving large sparse linear systems. PETSc, shorthand for the "Portable, Extensible Toolkit for Scientific computation," provides a variety of preconditioners and Krylov subspace solvers. Additive Schwarz methods (ASM) [10] derive a preconditioning by decomposing the problem domain into a number of possibly overlapping subdomains, (approximately) solving each subdomain, and summing the contributions of the subdomain solutions. Block-Jacobi preconditioning (BJacobi) [11] uses diagonal matrix blocks instead of diagonal matrix elements. Incomplete factorization (IC) [12] computes an incomplete factorization LU or Cholesky of the coefficient matrix and requires a solution of lower and upper triangular linear systems in every CG iteration. Parallel sparse approximate inverse (Parasails) [13], uses least-squares (Frobenius norm) minimization to compute a sparse approximate inverse. In addition, Hypre provides multigrid preconditioning for CG, boomerAMG [14], that constructs their multilevel hierarchy of solutions directly from the matrix coefficients, and they are simply subsets of unknowns without any geometric interpretation. Finally, a novel adaptive block factorized sparse approximate inverse nested with incomplete Cholesky (ABF-IC) preconditioning [15] is tested as a standalone CG code too.

Table 1
Key features of solver libraries.

Solver library	Direct/iterative	Algorithm	Parallel implementation
Mumps	Direct	Multifrontal	MPI
SuperLU	Direct	Supernodal	MPI
WSMP	Direct	Multifrontal	Hybrid (MPI/Pthreads)
ABF-IC	Iterative	PCG/(ABF-IC)	MPI
Hypre	Iterative	PCG/(ParaSails)	MPI
PETSc	Iterative	PCG (Bjacobi)	MPI

In addition to iterative solvers, three direct solver libraries were tested. The MUMPS (MUltifrontal Massively Parallel Solver) package is designed and developed by Amestoy et al. [16], and is a multifrontal code for solving both symmetric and unsymmetric systems and uses MPI at distributed memory systems. The Watson Sparse Matrix Package (WSMP) [17] was developed by Anshul Gupta of the IBM T. J.Watson Research Center. The package includes direct modified multifrontal solvers for both symmetric and unsymmetric systems. WSMP was primarily developed as a highly scalable parallel code and has a hybrid implementation with MPI and P-threads. SuperLU_DIST [18] is distributed memory (MPI) parallel version of the SuperLU family of solvers developed by Xiaoye (Sherry) Li of the University of California at Berkley. It is based on supernodal right looking LU factorization and is designed for general unsymmetric systems. Unlike symmetric solvers in MUMPS or WSMP, it computes and stores separate lower and upper triangular factors. In practice, it would not be typically used for SPD systems, which are the focus of this work, but we have included it to compare scalability of various direct factorization algorithms. All direct solvers in this study are in-core solvers; all factorization and solution data are kept in the computer's memory rather than written to temporary disk files. This makes these codes less dependent on the performance of file system, but also prone to failures if sufficient memory is not available.

The performance of preconditioned conjugate gradient and multigrid solvers in plane elasticity was compared by Jouglard et al. [19]. The difference between general sparse block solver and multifrontal solver was studied by Damhaug et al. [20]. Kilic et al. [21] showed that direct solvers provide faster solution then iterative solvers in implicit structural dynamics with ill-conditioned coefficient matrices. Two distributed memory solvers, MUMPS and SuperLU, were evaluated by Amestoy et al. [22]. Gould et al. [23] assessed the performance of direct solvers for symmetric matrices. The solvers were executed on a single processor for matrices of order greater than 10,000. Parallel performance of both direct and iterative solvers on proprietary IBM HPC platforms with matrices of 1–2 million unknowns were studied by Gupta et al. [24]. There is no general comparison of solvers on modern multi-core HPC clusters with many-million equations originating from practical 3D FEA discretization reported to date.

CAD models of solid geometries are commonly used to design parts and assemblies and create their corresponding part drawings for manufacturing. Conveniently, these models can be imported into FE packages for subsequent numerical analysis. While structured meshing with hexahedral elements exhibit higher convergence rate and accuracy, it frequently requires user intervention and is labor intensive. Automatic unstructured mesh generation with tetrahedral elements is quick and often preferred under a tight industrial project schedule. In biomechanics, however, finite element meshes generated from computed tomography (CT) [25] are almost always consisting of tetrahedral elements due to