Available online at www.sciencedirect.com
ScienceDirect
Comput. Methods Appl. Mech. Engrg. 284 (2015) 971–987
Computer methods in applied mechanics and engineering
www.elsevier.com/locate/cma

Computational cost of isogeometric multi-frontal solvers on parallel distributed memory machines

Maciej Woźniak[a], Maciej Paszyński[a,*], David Pardo[b,c,d], Lisandro Dalcin[e], Victor Manuel Calo[e,f]

[a] *AGH University of Sciences and Technology, Faculty of Computer Science, Electronics and Telecommunication, Department of Computer Science, al. A Mickiewicza 30, 30-059 Krakow, Poland*
[b] *Department of Applied Mathematics, Statistics, and Operational Research, University of the Basque Country (UPV/EHU), Bilbao, Spain*
[c] *Basque Center for Applied Mathematics (BCAM), Bilbao, Spain*
[d] *IKERBASQUE, Basque Foundation for Science, Bilbao, Spain*
[e] *King Abdullah University of Science and Technology, Center for Numerical Porous Media, Thuwal, Saudi Arabia*
[f] *King Abdullah University of Science and Technology, Applied Mathematics & Computational Science and Earth Science & Engineering, Thuwal, Saudi Arabia*

Available online 26 November 2014

Highlights

- We estimate computational cost of isogeometric solver on distributed memory parallel machines.
- We show p^2 scalability as we increase the global continuity, in 2D and 3D.
- We show O(N) and O($N^{4/3}$) costs for 2D and 3D parallel direct solvers.
- We verify the costs experimentally on STAMPEDE Linux cluster from TACC.
- We test MUMPS, PaSTiX, and SuperLU, through PETIGA toolkit built on top of PETSc.

Abstract

This paper derives theoretical estimates of the computational cost for isogeometric multi-frontal direct solver executed on parallel distributed memory machines. We show theoretically that for the C^{p-1} global continuity of the isogeometric solution, both the computational cost and the communication cost of a direct solver are of order $\mathcal{O}(\log(N)p^2)$ for the one dimensional (1D) case, $\mathcal{O}(Np^2)$ for the two dimensional (2D) case, and $\mathcal{O}(N^{4/3}p^2)$ for the three dimensional (3D) case, where N is the number of degrees of freedom and p is the polynomial order of the B-spline basis functions. The theoretical estimates are verified by numerical experiments performed with three parallel multi-frontal direct solvers: MUMPS, PaStiX and SuperLU, available through PETIGA toolkit built on top of PETSc. Numerical results confirm these theoretical estimates both in terms of p and N. For a given problem size, the strong efficiency rapidly decreases as the number of processors increases, becoming about 20% for 256 processors for a 3D example with 128^3 unknowns and linear B-splines with C^0 global continuity, and 15% for a 3D example with 64^3 unknowns and quartic B-splines with C^3 global continuity. At the same time, one cannot arbitrarily increase the problem size, since the memory required by higher order continuity spaces is large, quickly consuming all the available memory resources even in the parallel distributed memory version. Numerical results also suggest that the use of distributed parallel machines is highly beneficial

* Corresponding author. Tel.: +48 12 328 3314; fax: +48 12 617 5172.
E-mail address: maciej.paszynski@agh.edu.pl (M. Paszyński).

http://dx.doi.org/10.1016/j.cma.2014.11.020

when solving higher order continuity spaces, although the number of processors that one can efficiently employ is somehow limited.

Keywords: Multi-frontal direct solver; Isogeometric analysis; Parallel distributed memory machine; Computational cost; Communication cost

1. Introduction

In this paper, we focus on the distributed memory parallel solution of linear systems arising from the use of Isogeometric Analysis (IGA) [1] using direct solvers. IGA makes use of basis functions with high regularity (C^k with $k \geq 0$).

There exist two classes of methods for solving a linear system of equations: (a) direct methods, which deliver the exact solution in one step (up to round off error), and (b) iterative methods, which provide an approximate solution of prescribed quality by following an iterative process.

While the use of iterative solvers typically requires less computational resources (time and memory) than direct solvers, they suffer from a number of problems. First, iterative solvers often present severe convergence problems. Thus, different solvers are needed for each application (elasticity [2], electromagnetism [3], fluid dynamics [4]) and numerical methods. For IGA, various iterative solvers have been proposed in [5–9]. Second, in addition to the convergence problems, iterative solvers may be slower than direct solvers when a problem with multiple right-hand-side needs to be solved, as it occurs in the case of gradient-based inverse methods in order to compute the Jacobian and Hessian matrices. Iterative solvers may also be slower than direct solvers when several matrices with a common set of rows and columns need to be solved, as it occurs in mesh-based methods when local grid-refinements are performed [10–12]. Moreover, direct solvers are main building blocks of most iterative solvers. Thus, direct solvers become essential in many applications.

There exist several direct methods for the solution of a linear system of equations, including LU factorization, QR factorization, and singular value decomposition [13]. The fastest method is LU factorization, also known as Gaussian elimination, which is by far the most used algorithm for the direct solution of a system of linear equations. While other methods such as QR factorization may offer added stability minimizing the effect of round-off error, they are simply non-competitive in terms of computational efficiency. The main principle of the LU factorization algorithm is to decompose the original matrix A into the product of a lower triangular matrix L with an upper triangular matrix U.

For the case of sparse matrices, it is important to avoid operations with the zeros of the matrix, and to produce L and U factors that are as sparse as possible. State-of-the-art implementations of the LU factorization algorithm for sparse matrices include the frontal [14,15] and multi-frontal solvers [16,17]. The latest trends on this area include efficient parallelization techniques (see e.g., [18,19]) and application-specific implementations that take advantage of the data-structures of the Galerkin method, such as the works of [20–24].

This paper derives theoretical estimates of the computational cost for isogeometric (IGA) multi-frontal direct solver executed on distributed memory parallel machines, for 1D, 2D and 3D problems. We show that for the C^{p-1} global continuity of the solution, the computational cost of the parallel solver executed on a distributed memory cluster is of order $\mathcal{O}(\log(N)p^2)$ for the 1D case, $\mathcal{O}(Np^2)$ for the 2D case, and $\mathcal{O}(N^{3/4}p^2)$ for the 3D case. These theoretical estimates are compared with the ones obtained for the sequential multi-frontal direct solver described in [25] as well as for the shared memory parallel solver for GPU described in [21]. Namely, the sequential estimates show that the computational cost of C^{p-1} global continuity of the isogeometric solution is of order $\mathcal{O}(Np^3)$ for the 1D case, $\mathcal{O}(N^{1.5}p^3)$ for the 2D case, and $\mathcal{O}(N^2p^3)$ for the 3D case. In particular, the computational cost of sequential IGA direct solvers grows as p^3 when we increase the global continuity. In [21], we already showed that we can reduce this p^3 factor down to p^2 when using a parallel shared memory machine. Our parallel distributed memory direct solver delivers the same computational complexity as the shared memory parallel solver [21], for a sufficiently large number of cores. Its communication complexity is of the same order as the computational complexity.

We confirm our theoretical estimates with numerical experiments, quantifying the weak and strong scalability of the direct solver for IGA. The theoretical estimates concerning the computational costs are compared with numerical experiments performed on the STAMPEDE Linux cluster from the Texas Advanced Computing Center [26]. The experiments are performed with the PETIGA toolkit [27] built on the PETSc library [28–30], and utilize the parallel