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

MUMPS solver [31,32,18] with parallel Scalapack dense solver [33], the parallel SuperLU solver [34,35], and parallel PaStiX solver [36].

2. Model problem

In this section, we describe our model problem. We focus on the conductive media equation

$$-\nabla \cdot \sigma \nabla u = \nabla \cdot \mathbf{J}^{imp}, \tag{1}$$

where σ is the conductivity of the media, u is the electric potential, and $\mathbf{J}^{imp}$ is the impressed electric current (the source). The above partial differential equation (PDE) is imposed on a computational domain $\Omega = [0, 1]^d$, where d is the spatial dimension. We impose homogeneous Dirichlet boundary conditions

$$u = 0 \quad \text{on } \Gamma_D = \partial \Omega. \tag{2}$$

The weak variational formulation is obtained by taking the L^2-scalar product with functions $v \in H_0^1(\Omega) = \{v \in H^1(\Omega) : v|_{\Gamma_D} = 0\}$, integrating by parts, and imposing the Dirichlet boundary conditions:

$$\text{Find } u \in V = H_0^1(\Omega) \text{ such that} \tag{3}$$
$$b(v, u) = l(v), \quad \forall v \in V, \tag{4}$$

where

$$b(v, u) = \int_\Omega \sigma \nabla v \cdot \nabla u dx, \quad \text{and} \tag{5}$$
$$l(v) = \int_\Omega v \cdot \nabla \cdot \mathbf{J}^{imp} dx. \tag{6}$$

In order to make the estimation of the computational complexity of the multi-frontal solver tractable,

- we assume that the grid is regular, and it has the same number of degrees of freedom in each coordinate direction;
- we assume that the polynomial order of approximation is constant throughout the entire grid;
- we assume that the number of elements is sufficiently large;
- we ignore possible orthogonality relationships between basis functions, treating as zero only those matrix contributions coming from basis functions with disjoint supports (the so called "logical" zeros);
- we only consider the limiting case of C^{p-1} continuity; and
- we restrict our attention to the Laplace problem in the unit tensor product domain.

The computational cost of the solver is independent of the considered PDE as long as it is given by a single scalar equation in $H^1(\Omega)$. A different equation would modify the values in the frontal matrices, but the location of the logical zeros would remain unaffected.

The computational cost of the solver is also independent of geometrical variations. The geometry may affect the value of the Jacobian in the integrals, but not the location of the nonzero entries. Our study, however is limited to a single patch of elements. In a case of multiple clusters, the solver should return the Schur complements with respect to the boundary of each patch being used, and the Schur complements must be processed on a higher level by some external solver.

3. Theoretical complexity estimates for direct solvers in arbitrary dimension

In this section, we derive estimates for the number of FLOPs required to solve a system of linear equations using a direct multi-frontal solver on a distributed memory parallel machine for one, two, and three dimensional problems.

3.1. Schur complement

We first analyze the FLOPs required to perform the Schur complement operation, which is the main building block for construction of a multi-frontal solver. The number of floating point operations can be estimated by noticing that computing the Schur complement is equivalent to the execution of a partial forward elimination.

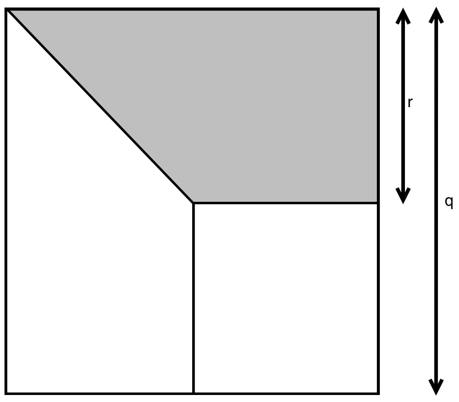


Fig. 1. Visual explanation of q and r.

In order to estimate the number of FLOPs, we define variable q as the number of fully assembled degrees of freedom that can be eliminated from an arbitrary frontal matrix M, and r as the number of not-fully assembled degrees of freedom that form a Schur complement sub-matrix (see Fig. 1). Thus, the total number of operations needed for the partial elimination $S(q,r)$ is equal to:

$$S(q,r)=\sum_{m=1}^{q}3(m+r)^2. \tag{7}$$

The above identity is obtained by eliminating q rows from a matrix with size $(q+r)$. The elimination of the first row involves $(q+r)^2$ subtractions, the elimination of the second row involves $(q+r-1)^2$ subtractions, and so on, up to the last row to be eliminated, which involves $(r+1)^2$ subtractions. The exact number of operations involves $3(m+r)^2$ operations instead of $(m+r)^2$ operations, since for each entry we perform a multiplication, a division and a subtraction.

In this paper we focus on parallel distributed memory machines, and we assume that we use one core per processor, with each processor having its own local memory. Under such assumption, we can use the term "cores" as equivalent to the term "processors". We also assume that we have enough cores available to perform all the row subtractions concurrently, thus we can reduce the sequential cost $S(q,r)$ to concurrent cost $C(q,r)$ like

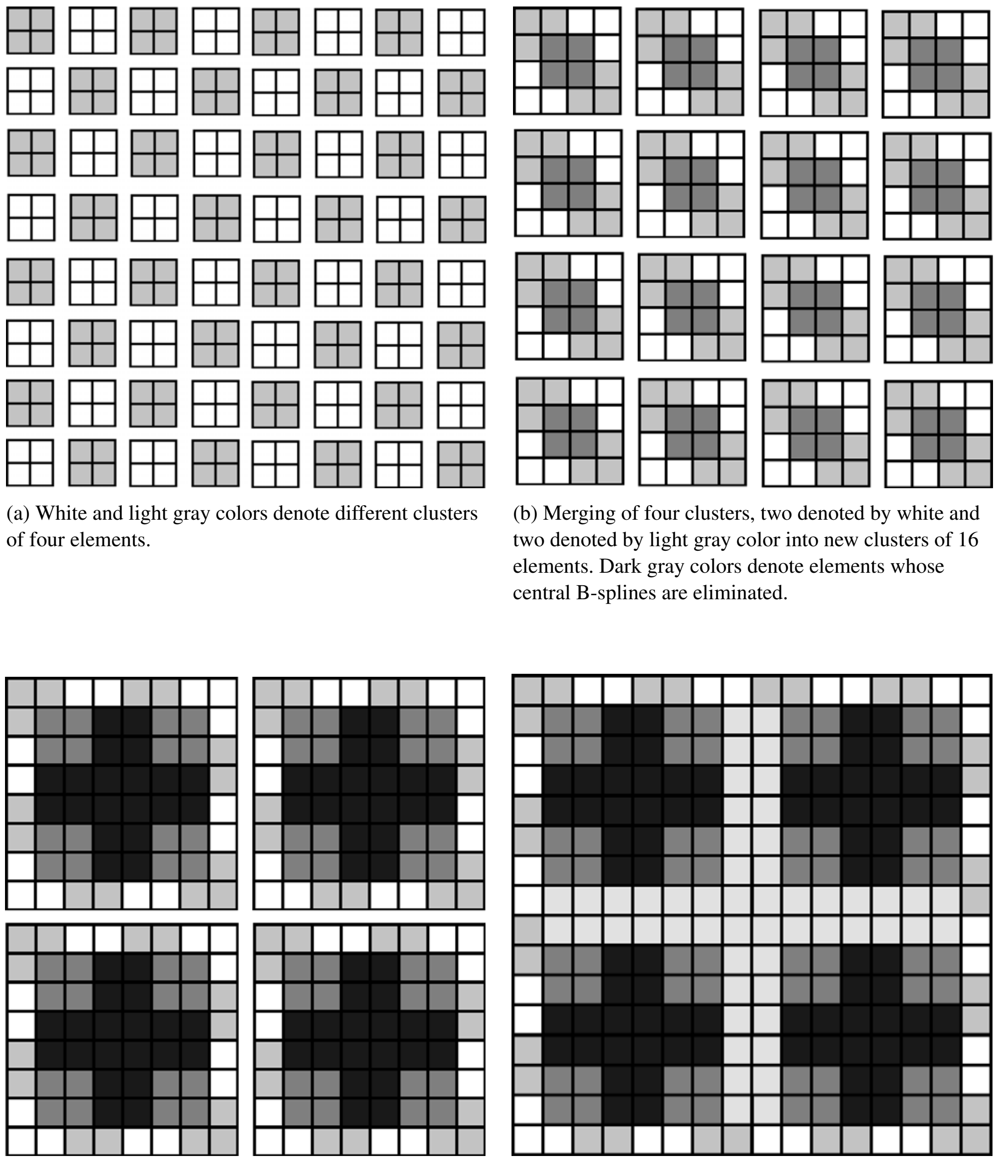
$$C(q,r)=\sum_{m=1}^{q}3(m+r)=3\frac{((r+1)+(r+q))q}{2}=O\left(q^2+qr\right). \tag{8}$$

3.2. The multi-frontal solver

We divide our computational domain in N_p clusters of elements. For IGA, each patch is a set of p consecutive elements in each dimension. For simplicity, we assume that the number of clusters of our computational domain is $(2^s)^d$, where s is an integer, and d is the spatial dimension of the problem. Even when this assumption is not satisfied, the final limiting result still holds true.

The idea of the multi-frontal solver is to eliminate interior unknowns of each patch, then join each 2^d clusters into one to produce $(2^{s-1})^d$ new clusters, eliminate interior unknowns of each new patch, and continue with the iterative procedure until the last 2^d clusters are joined into one. The sequential iterative algorithm can be expressed as follows:

```
1   for $i = 0,\ s-1$:
2     $N_p = N_p(i) = (2^{s-i})^d$
3     if $i = 0$, define $N_p(0)$ clusters.
4     else join old $N_p(i-1)$ clusters to define $N_p(i)$ new clusters.
5     endif
6     Eliminate interior unknowns of each patch.
7   end for
8   Solve dense boundary problem.
```


(a) White and light gray colors denote different clusters of four elements.

(b) Merging of four clusters, two denoted by white and two denoted by light gray color into new clusters of 16 elements. Dark gray colors denote elements whose central B-splines are eliminated.

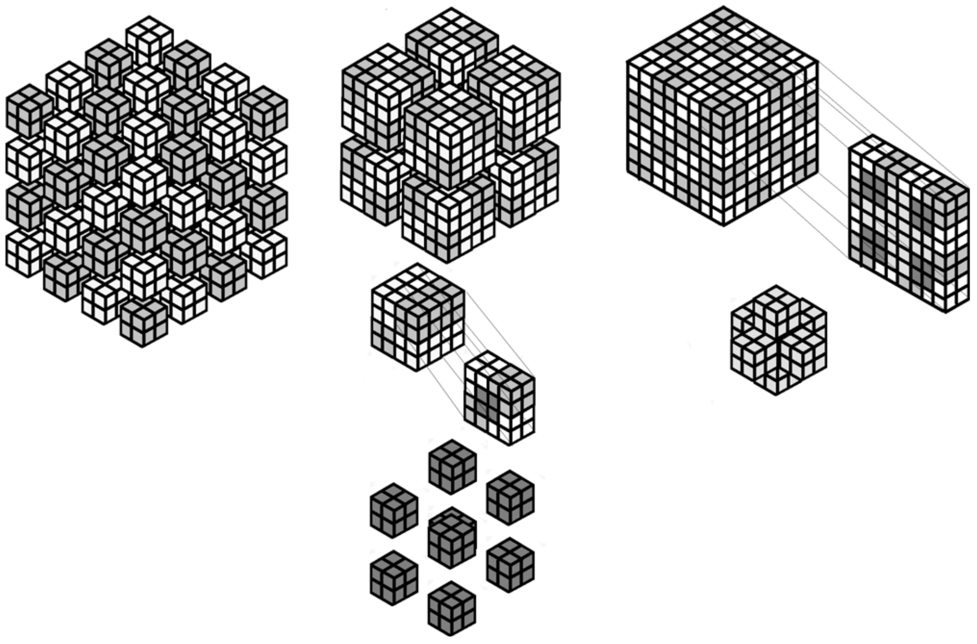
(c) Merging of four clusters, into new clusters of 64 elements. Middle-gray color denotes elements whose B-splines have been already eliminated. Dark gray color denotes elements whose B-splines are eliminated at this step.

(d) Merging of four clusters into new clusters of 256 elements. Central light gray color denotes elements whose B-splines are eliminated at this step.

Fig. 2. The scheme of the multi-frontal solver algorithm execution over a 2D grid for quadratic B-splines. Each element contains the entire support of one B-spline with its maximum value attained at its center.

The algorithm is illustrated in Fig. 2 for the 2D case, with C^1 quadratic B-splines. In this example, we consider $(2^s)^d$ with $s = 3$ and $d = 2$, that is $(2^3)^2 = 64$ clusters, each one with $p^d = 2^2 = 4$ elements.

- In the first step presented on top-left panel in Fig. 2, we define $N_p(0) = (2^3)^2 = 64$ clusters, and nothing is eliminated in this step.
- In the second step depicted on the top-right panel in Fig. 2, we merge sets of four clusters from the previous step to create $N_p(1) = (2^2)^2 = 16$ clusters, and we eliminate four basis functions from the interior (denoted in Fig. 2 with gray color).



(a)
White and light gray colors denote different clusters of eight elements.

(b)
Merging of eight clusters, four denoted by white and four denoted by light gray color into new clusters of 64 elements. Dark gray colors denote elements whose central B-splines are eliminated.

(c)
Merging of eight clusters into new clusters of 512 elements. Dark gray color denotes elements whose B-splines have been already eliminated. Light gray color denotes a 3D cross with elements whose B-splines are eliminated at this step.

Fig. 3. The scheme of the multi-frontal solver algorithm execution over a 3D grid for quadratic B-splines. Each element contains the entire support of one B-spline with its maximum value attained at its center.

- In the third step described in the bottom-left panel in Fig. 2, we join sets of four clusters from the previous step to create $N_p(2) = (2^1)^2 = 4$ clusters, and we eliminate 20 basis functions from the interior of each patch, forming a cross inside each patch (denoted in Fig. 2 by dark gray color).
- In the fourth step displayed on the bottom-right panel of Fig. 2, we merge sets of four clusters from the previous step to create $N_p(2) = (2^0)^2 = 1$ patch, we eliminate 13*4 basis functions from the interior of the patch (forming a 2D cross, denoted in Fig. 2 by light gray color), and we are left with an interface problem.
- To conclude, we solve the dense interface problem.

The algorithm is also illustrated in Fig. 3 for the 3D case, also with quadratic B-splines. In this example, we have $(2^s)^d = (2^2)^3 = 64$ clusters, each one with $p^d = 2^3 = 8$ elements.

As derived in [25], the number of FLOPs required by the above algorithm in the sequential version can be expressed as:

$$\sum_{i=0}^{s-1} N_p(i) \cdot S(i), \tag{9}$$

where $S(i)$ is the cost (FLOPs) of performing each Schur complement at the ith step, and $s = \log_2(N^{1/d})$. Following the notation of the previous subsection on the Schur complement, we define $q = q(i)$ as the number of interior unknowns of each path at the ith step, and $r = r(i)$ as the number of interacting unknowns at the ith step (see Table 1).

Table 1
Number of interior and interacting unknowns at each step of the multi-frontal solver.

	$q(0)$	$r(0)$	$q(i),\ i \neq 0$	$r(i),\ i \neq 0$	$N_p(i)$	$S(i)$	$C(i)$
IGA C^{p-1}	$\mathcal{O}(1)$	$\mathcal{O}(p^d)$	$\mathcal{O}(2^{(d-1)i}p^d)$	$\mathcal{O}(2^{(d-1)i}p^d)$	$\mathcal{O}((2^{s-i})^d)$	$r(i)^3$	$r(i)^2$

In order to estimate the computational and communication costs for the parallel distributed memory multi-frontal solver, we further assume that:

- there is a sufficiently large number of processors available,
- a sufficiently large amount of memory is available for each core
- the amount of data exchanged during the communication in the parallel multi-frontal solver is dominated by the size of the Schur complement matrices to be exchanged between processors.

Under the above assumptions, the computational cost for the parallel solver can be estimated as

$$\sum_{i=0}^{s-1} C(i) t_{comp}, \tag{10}$$

where t_{comp} denotes the time of performing a single FLOP operation, $C(i)$ is the cost (FLOPs) required to perform concurrent Schur complement computations at the ith step, with concurrent row subtractions executed simultaneously for all the matrices from the current ith step.

Moreover, the communication cost can be estimated as

$$\sum_{i=0}^{s-1} \left(t_{init} + C(i) t_{comm}\right), \tag{11}$$

where t_{init} denotes the initialization time of a single message, and t_{comm} denotes the time of communicating single floating point data, and the amount of exchanged data is proportional to the number of entries in the matrix, which in turn is equal to $C(i)$.

We can estimate now the computational and communication complexities for both sequential and parallel version in the following way

- 1D IGA:

$$\text{FLOPs} = 2^s p^2 + \sum_{i=1}^{s-1} 2^{s-i} p^3 = \mathcal{O}(2^s p^3) = \mathcal{O}(N p^3),$$

$$\begin{aligned} \text{Parallel cost} &= p^2 t_{comp} + \sum_{i=1}^{s-1} p^2 t_{comp} + \sum_{i=1}^{s-1} \Big(t_{init} + p^2 t_{comm}\Big) \\ &= \mathcal{O}(s p^2 t_{comp} + s t_{init} + s p^2 t_{comm}) \\ &= \mathcal{O}(\log(N) p^2 t_{comp} + \log(N) t_{init} + \log(N) p^2 t_{comm}). \end{aligned} \tag{12}$$

- 2D IGA:

$$\begin{aligned} \text{FLOPs} &= 2^{2s} p^4 + \sum_{i=1}^{s-1} 2^{2(s-i)} 2^{3i} p^6 = \mathcal{O}(2^{2s} p^4 + 2^{3s} p^6) \\ &= \mathcal{O}(N_p^3 p^6) = \mathcal{O}(N^{1.5} p^3) \end{aligned}$$

$$\begin{aligned} \text{Parallel cost} &= p^4 t_{comp} + \sum_{i=1}^{s-1} 2^{2i} p^4 t_{comp} + \sum_{i=1}^{s-1} \Big(t_{init} + 2^{2i} p^4 t_{comm}\Big) \\ &= \mathcal{O}(p^4 t_{comp} + 2^{2s} p^4 t_{comp} + s t_{init} + 2^{2s} p^4 t_{comm}) \\ &= \mathcal{O}(N p^2 t_{comp} + \log(N^{0.5}) t_{init} + N p^2 t_{comm}). \end{aligned} \tag{13}$$

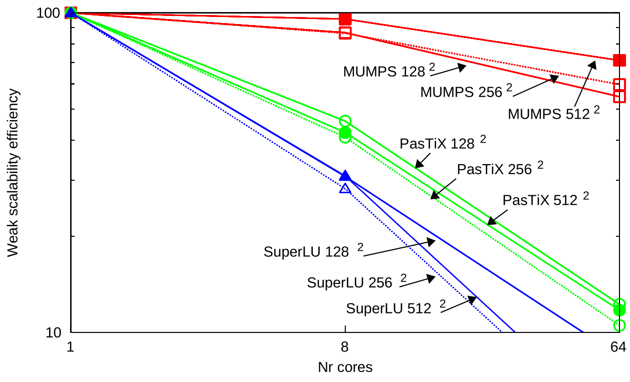


Fig. 4. Weak scaling efficiency of the direct solvers for 2D IGA with linear B-splines, C^0 global continuity.

- 3D IGA:

$$\begin{aligned}
\text{FLOPs} &= 2^{3s}p^6 + \sum_{i=1}^{s-1} 2^{3(s-i)}2^{6i}p^9 = \mathcal{O}(2^{3s}p^6 + 2^{6s}p^9) \\
&= \mathcal{O}(N_p^3 p^6 + N_p^6 p^9) = \mathcal{O}(N^2 p^3) \\
\text{Parallel cost} &= p^6 t_{comp} + \sum_{i=1}^{s-1} 2^{4i}p^6 t_{comp} + \sum_{i=1}^{s-1}\left(t_{init} + 2^{4i}p^6 t_{comm}\right) \\
&= \mathcal{O}(p^6 t_{comp} + 2^{4s}p^6 t_{comp} + s t_{init} 2^{4s} p^6 t_{comm}) \\
&\quad - \mathcal{O}(N^{4/3}p^2 t_{comp} + \log(N^{1/3}) t_{init} + N^{4/3} p^2 t_{comm}). \qquad (14)
\end{aligned}$$

4. Numerical results

In this section, we present numerical experiments to verify the theoretical estimates presented in the previous section. The numerical experiments are performed on STAMPEDE [26], a Linux cluster hosted by the Texas Advanced Computing Center. The simulations are performed with PETIGA [27], and utilize parallel MUMPS solver [31,32,18] with parallel Scalapack [33] dense solver, parallel SuperLU solver [34,35], and parallel PaStiX solver [36] available from PETSc library [28–30]. In the experiments, we utilized one core per Linux cluster node in order to maximize the amount of available memory per node. The only exception is the two experiments with 512 cores for linear B-splines in 3D, where we utilize 2 cores per node.

4.1. 2D case

The experiments have been performed for different mesh sizes, namely 128^2, 256^2, 512^2, 1024^2 and 2048^2 elements, and for a variable number of processors up to 256. Basically, it is not possible to solve any larger problem for 2D IGA on STAMPEDE with direct solvers, since all the solvers run out of memory. We could increase the problem size by employing out of core capabilities. However, in such case the computational cost estimates would not be representative of the process, since the cost is limited by the performance of the parallel file system [37].

4.1.1. Weak scaling efficiency

We start illustrating the weak scaling efficiency for the three different direct solvers executed for 2D IGA problem. The weak efficiency is computed using the formula $E^{weak} = \frac{T_1^{weak}}{T_c^{weak}} * 100$, where T_1^{weak} denotes the execution time of a single core processing a single workload, and T_c^{weak} is the execution time of c cores processing c workloads, one workload per core. The weak scaling efficiency results are presented in Figs. 4–6, for MUMPS, PasTiX and SuperLU, for linear, quartic and octic B-splines, with C^0, C^3 and C^7 global continuity, respectively.

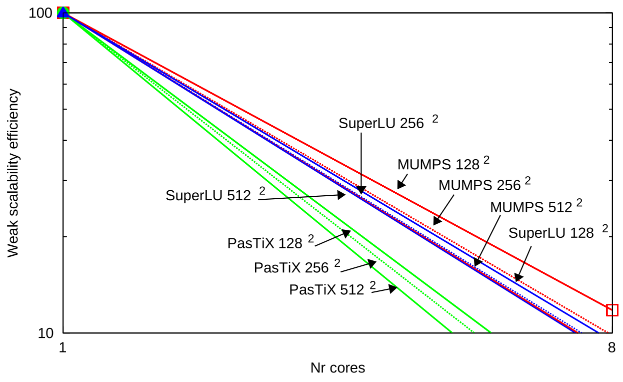


Fig. 5. Weak scaling efficiency of the direct solvers for 2D IGA with quartic B-splines, C^3 global continuity.

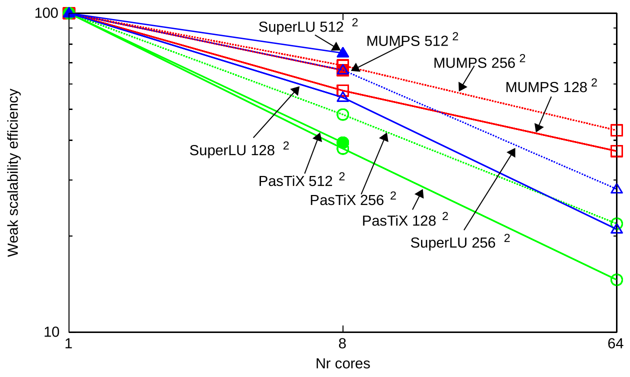


Fig. 6. Weak scaling efficiency of the direct solvers for 2D IGA with octic B-splines, C^7 global continuity.

We can draw the following conclusions:

- From Fig. 4, MUMPS solver for linear B-splines and C^0 continuity attains 70% weak efficiency up to 64 cores.
- Efficiency of PasTiX and SuperLU for linear B-splines and C^0 continuity decreases down to approx. 10% for 64 cores.
- MUMPS solver for quartic B-splines and C^3 continuity attains XX percent weak efficiency up to 32 cores. For 64 cores, it runs out of memory.
- Efficiency of PasTiX and SuperLU for linear B-splines and C^3 continuity decreases down to approx. XX percent for 32 cores. For 64 cores, they run out of memory.
- MUMPS solver for octic B-splines and C^7 continuity attains 40% weak efficiency up to 64 cores.
- Efficiency of PasTiX and SuperLU for octic B-splines and C^7 continuity decreases down to 15%–25% for 64 cores.

4.1.2. *Strong scaling efficiency*

Let us focus now on strong scaling efficiency of the MUMPS solver executed for 2D IGA problem. The strong scaling efficiency is computed based on formula $E^{strong} = \frac{T_1^{strong}}{c*T_c^{strong}} * 100$ where T_1^{strong} denotes the execution time of a single core processing workload of size N, and T_c^{strong} is the execution time of c cores still processing the workload of size N, now distributed into c processors.

The strong scaling efficiency results are presented in Fig. 7, for linear, quartic and octic B-splines, with C^0, C^3 and C^7 global continuity, respectively. The experiments have been performed for different mesh sizes, namely 128^2, 256^2, 512^2, 1024^2 and 2048^2 elements.

From the presented results, we conclude the following:

- The larger the problem, the better the scalability.
- The larger the problem, the more difficult is to fit it in memory.

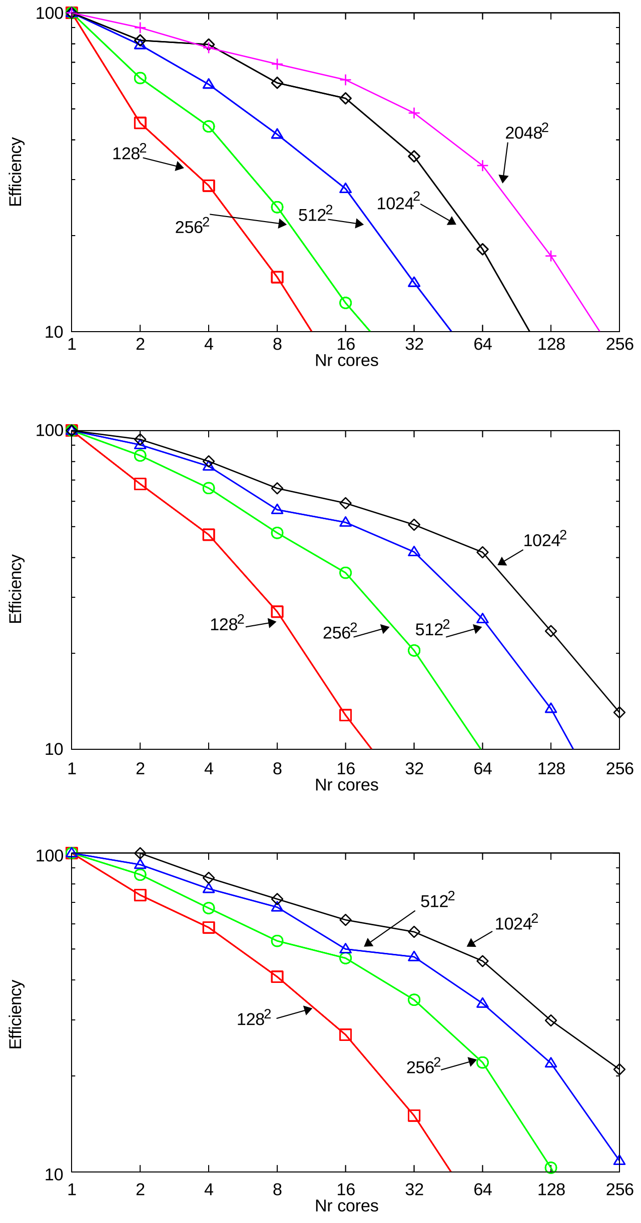


Fig. 7. Parallel efficiency of the MUMPS direct solver for 2D IGA with linear (top panel), quartic (middle panel) and octic (bottom panel) B-splines, with C^0, C^3 and C^7 global continuity, respectively.

- Quite rapidly (128–256 cores), the scalability is limited by the combination of both factors. If the problem is too large, then it does not fit in memory. Otherwise, the number of processors for which it properly scales is also limited.
- We observe better strong scalability for C^7 global continuity with octic B-splines than for C^0 global continuity with linear B-splines.

4.2. $\mathcal{O}(Np^2)$ cost

Figs. 8 and 9 illustrate the parallel scalability of MUMPS, PaStiX and SuperLU solvers executed on 8 and 32 nodes, one core per node. They display the execution time divided by a p^2 factor. Thus, a horizontal line represents the ideal p^2 growth of the solver, a descending line denotes a growth better than p^2, and an ascending line denotes a

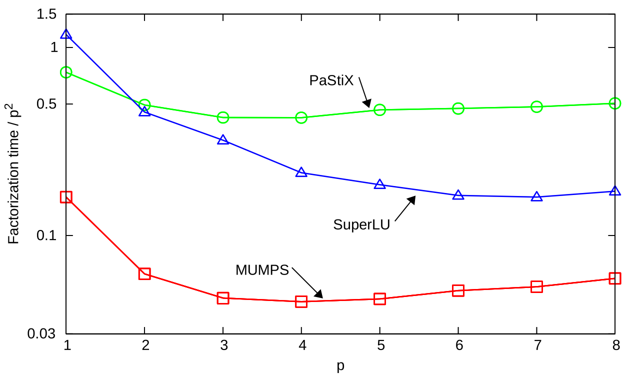


Fig. 8. Execution time divided by p^2 measured for parallel MUMPS, SuperLU and PaStiX solvers executed over distributed memory machine, for a two dimensional problem with 256 × 256 elements, for continuity C^{p-1} for different p, one core per node, with eight nodes.

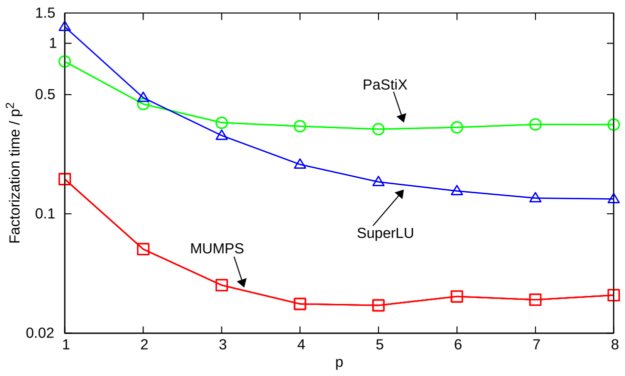


Fig. 9. Execution time divided by p^2 measured for parallel MUMPS, SuperLU and PaStiX solvers executed over distributed memory machine, for a two dimensional problem with 256 × 256 elements, for continuity C^{p-1} for different p, one core per node, with thirty two nodes.

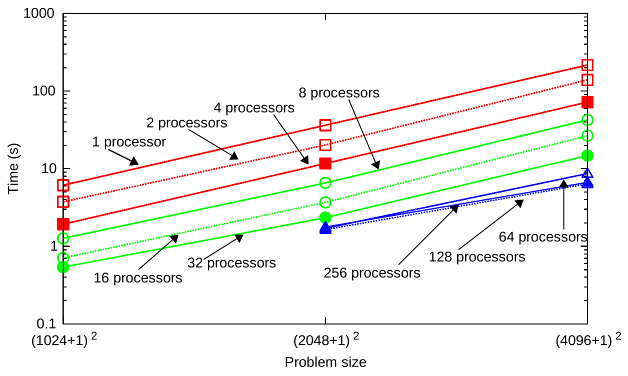


Fig. 10. Execution times for the MUMPS direct solver for 2D IGA with linear B-splines, C^0 global continuity.

growth worse than p^2. As predicted by our model, the solvers scale like p^2, especially when we increase the number of processors to 32.

Finally, we display the execution times of the parallel MUMPS solver for the 2D IGA model problem, for increasing problem size N and for fixed p, and execute the curve fitting algorithm to estimate the exponent factor in formula const $* N^{\alpha}$. The execution times as a function of N are presented in Fig. 10 for linear B-splines and C^0 global continuity; in Fig. 11 for quartic B-splines and C^3 global continuity; in Fig. 12 for octic B-splines and C^7 global continuity.

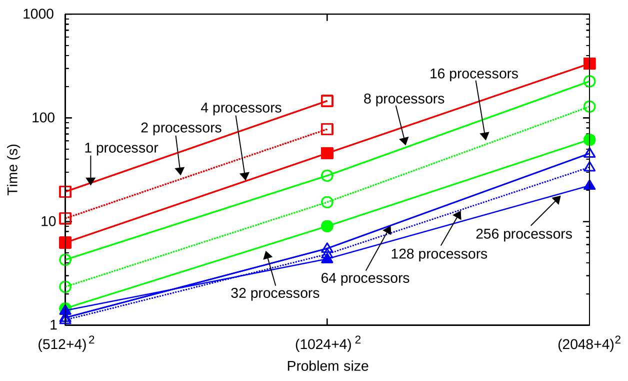


Fig. 11. Execution times for the MUMPS direct solver for 2D IGA with quartic B-splines, C^3 global continuity.

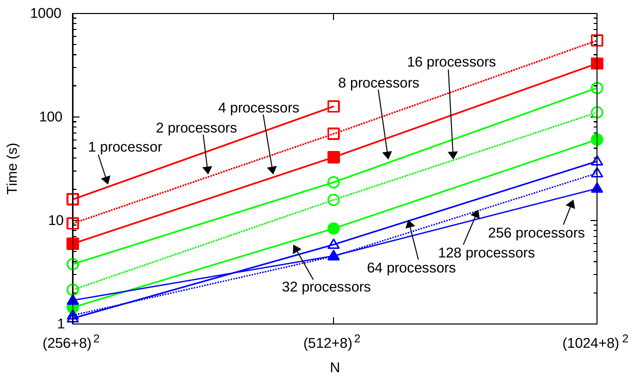


Fig. 12. Execution times for the MUMPS direct solver for 2D IGA with octic B-splines, C^7 global continuity.

Table 2
Exponent factors α from fitting the curve const $* N^\alpha$ based on execution times of MUMPS solver for 2D IGA with linear B-splines, C^0 continuity.

Nr_{cores}	1	2	4	8	16	32	64	128	256
α	1.2847	1.3057	1.3066	1.266	1.3071	1.1922	1.1712	0.9771	0.9562

The curve fitting algorithm estimated the α exponent factor as summarized in Table 2 for linear B-splines, in Table 3 for quartic B-splines and in Table 4 for octic B-splines, all with C^{p-1} global continuity. In all cases, the exponent factor converges to 1, which results in linear $\mathcal{O}(N)$ computational cost for fixed p, as predicted by the theory.

4.3. 3D case

4.3.1. Weak scaling efficiency

We start illustrating the weak scaling efficiency of MUMPS solver executed for 3D IGA problem. The weak scaling efficiency results are presented in Figs. 13 and 14 for linear and quartic B-splines, with C^0 and C^3 global continuity, respectively. The experiments have been performed for different mesh sizes, namely 16^3, 32^3, 64^3 and 128^3 elements.

For large workloads (32^3 elements per core), the weak scaling efficiency is around 35% for 8 processors and less than 10% for 64 processors for linear B-splines with C^0 continuity; around 10% for 8 processors for quartic B-splines with C^3 continuity. For octic B-splines with C^7 continuity we cannot present weak scalability since we run out of memory.

Table 3
Exponent factors α from fitting the curve const $* N^{\alpha}$ based on execution times of MUMPS solver for 2D IGA with quartic B-splines, C^3 continuity.

Nr_{cores}	1	2	4	8	16	32	64	128	256
α	1.4256	1.4373	1.4404	1.4361	1.4503	1.3574	1.323	1.2286	1.0049

Table 4
Exponent factors α from fitting the curve const $* N^{\alpha}$ based on execution times of MUMPS solver for 2D IGA with octic B-splines, C^7 continuity.

Nr_{cores}	1	2	4	8	16	32	64	128	256
α	1.5232	1.4925	1.4692	1.4385	1.4477	1.3693	1.3519	1.3498	1.0937

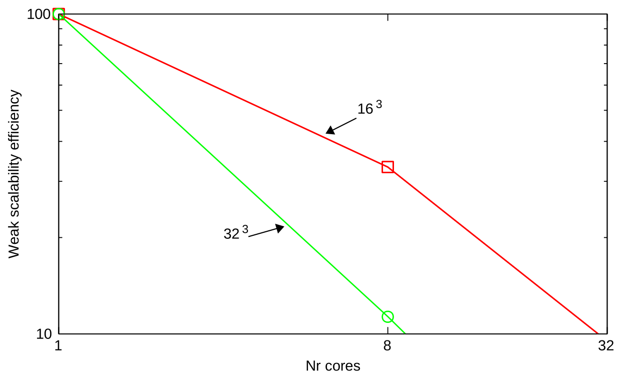


Fig. 13. Weak scalability of the MUMPS direct solver for 3D IGA with linear B-splines, C^0 global continuity. Different lines correspond to various problem sizes.

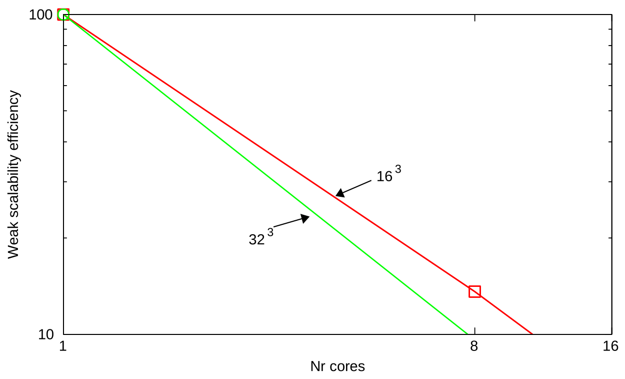


Fig. 14. Weak scalability of the MUMPS direct solver for 3D IGA with quartic B-splines, C^3 global continuity. Different lines correspond to various problem sizes.

4.3.2. *Strong scaling efficiency*

Let us focus now on strong scaling efficiency of the MUMPS solver executed for 3D IGA problem. The efficiency results are presented in Figs. 15 and 16, for linear and quartic B-splines, with C^0 and C^3 global continuity, respectively. The experiments have been performed for different mesh sizes, namely 16^3, 32^3, 64^3 and 128^3 elements.

From the presented results we conclude the following:

- The larger the problem, the better the scalability.
- The larger the problem, the more difficult is to fit it in memory.

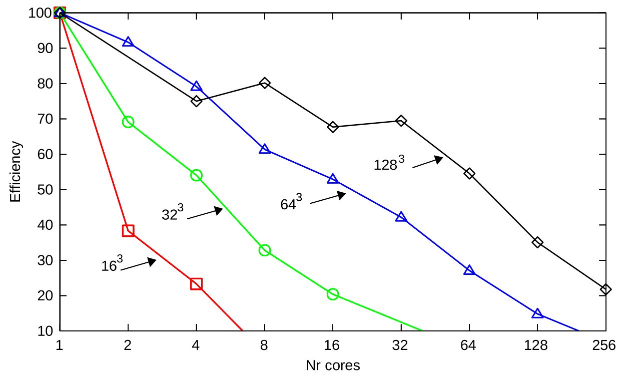


Fig. 15. Parallel efficiency of the MUMPS direct solver for 3D IGA with linear B-splines, C^0 global continuity. Different lines represent different sizes of the mesh.

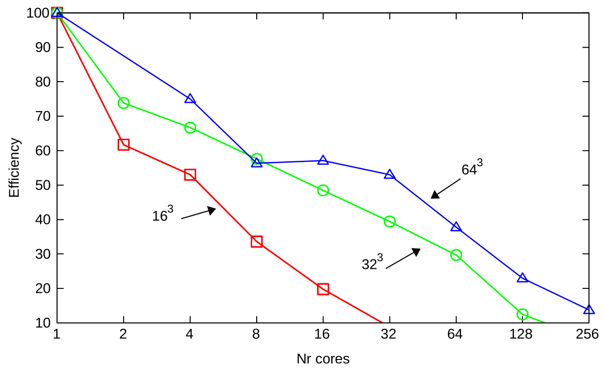


Fig. 16. Parallel efficiency of the MUMPS direct solver for 3D IGA with quartic B-splines, C^3 global continuity. Different plots represent different sizes of the mesh.

- Quite rapidly (128–256 cores), the scalability is limited by the combination of both factors. If the problem is too large, then it does not fit in memory. Otherwise, the number of processors for which it properly scales is also limited.
- We observe better scalability for C^3 global continuity with quartic B-splines than for C^0 global continuity with linear B-splines.

4.4. $\mathcal{O}(N^{4/3}p^2)$ cost

We display the execution times of the parallel MUMPS solver for the 3D IGA model problem, as we increase problem size N for fixed p, and execute the curve fitting algorithm to estimate the exponent factor in const $* N^{\alpha}$. The execution times as a function of $N^{1/3}$ are presented in Fig. 17 for linear B-splines with C^0 global continuity; in Fig. 18 for quartic B-splines with C^3 global continuity.

The curve fitting algorithm estimated the α exponent factor as summarized in Table 5 for linear B-splines, and in Table 6 for quartic B-splines, all with C^{p-1} global continuity. We do not display the curve fitting on the octic B-splines case, because we do not have enough data. In these two cases, the exponent factor converges to $4/3$, which results in $\mathcal{O}(N^{4/3})$ computational cost for fixed p, as predicted by theory.

Finally, we display the execution times of the parallel MUMPS solver for the 3D IGA model problem, for increasing continuity p and fixed N, divide the execution time by $N^{4/3}$, and execute the curve fitting algorithm to estimate the exponent factor in formula $\mathcal{O}\left(\text{const} * p^{\beta}\right)$. The execution times as a function of p are presented in Fig. 19 for quartic B-splines, C^3 global continuity.

The curve fitting algorithm estimated the β exponent factor as summarized in Table 7 for quartic B-splines with C^3 global continuity. The exponent factor converges to 2, which results in $\mathcal{O}(p^2)$ growth for fixed N, as predicted by the theory.

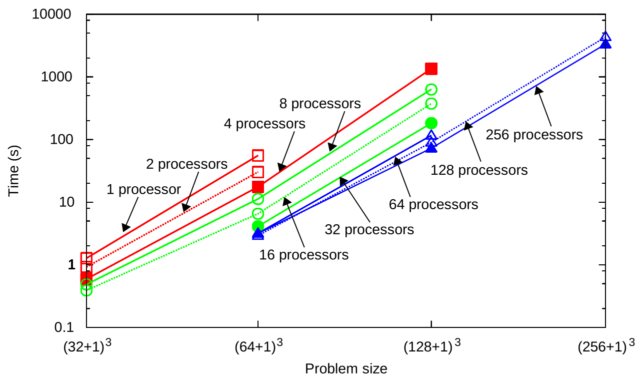


Fig. 17. Execution times for the MUMPS direct solver for 3D IGA with linear B-splines, C^0 global continuity.

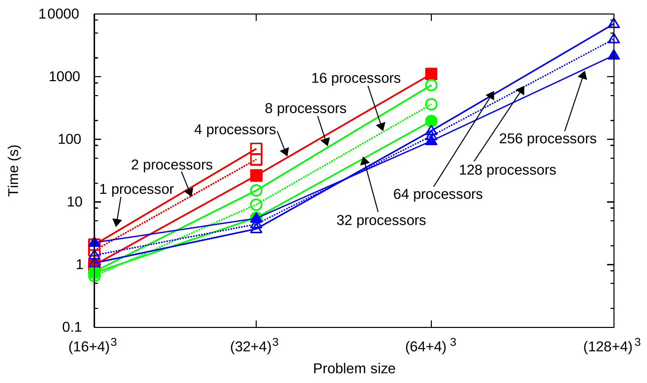


Fig. 18. Execution times for the MUMPS direct solver for 3D IGA with quartic B-splines, C^3 global continuity.

Table 5
Exponent factors α from fitting the curve const $* N^{\alpha}$ based on execution times of MUMPS solver for 3D IGA with linear B-splines, C^0 continuity.

Nr_{cores}	1	2	4	8	16	32	64	128	256
α			1.8596	1.7234	1.6501	1.8237	1.7273	1.7573	1.6738

Table 6
Exponent factors α from fitting the curve const $* N^{\alpha}$ based on execution times of MUMPS solver for 3D IGA with quartic B-splines, C^3 continuity.

Nr_{cores}	1	2	4	8	16	32	64	128	256
α		2.1818	2.1309	2.0827	1.9195	1.7393	1.7002	1.5704	1.364

5. Conclusions

In this paper, we analyzed theoretically and experimentally the performance of multi-frontal direct solvers on distributed memory parallel machines. The theoretical estimates assume sufficiently large number of processors to perform concurrent row subtractions during the local factorizations. We show that the computational cost of the parallel direct solvers grows as p^2 when increasing the global continuity and N is fixed. Additionally, for fixed p, we show that the 2D parallel direct solver delivers linear computational and communication costs. In 3D, the computational and communication costs of the parallel solvers grow in terms of the problem size N as $O(N^{4/3})$ when executed on distributed memory parallel machines. The obtained computational cost estimates for the distributed memory parallel direct solver are similar to those obtained for shared memory parallel machines [21]. The difference

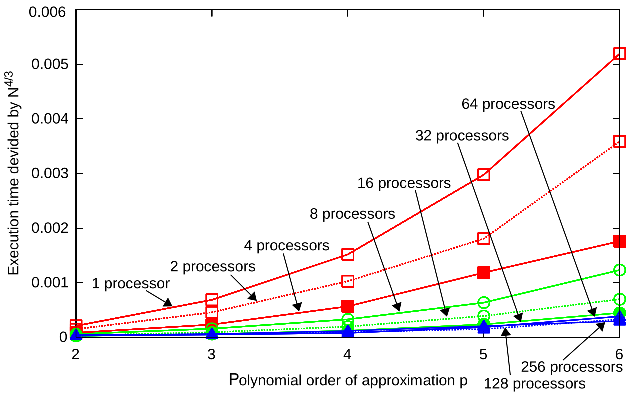


Fig. 19. Execution times for the MUMPS direct solver for 3D IGA, for fixed 32^3 elements, divided by $N^{4/3}$.

Table 7
Exponent factors β from const $* p^\beta$ curve fitting based on execution times of MUMPS solver for 3D IGA with quartic B-splines, C^3 continuity, for fixed N, divided by $N^{4/3}$.

Nr_{cores}	1	2	4	8	16	32	64	128	256
β	2.905	2.849	2.8429	2.7434	2.7818	2.6344	2.3794	2.0905	1.8205

in the derivation is that here it appears an additional term related to the communication cost. The theoretical estimates are verified with numerical experiments.

Acknowledgments

The work of MP and MW was supported by Polish National Science Centre grant no. 2012/07/B/ST6/01229 and 2012/06/M/ST1/00363. The work of DP was partially funded by the Project of the Spanish Ministry of Economy and Competitiveness with reference MTM2013-40824-P, the BCAM Severo Ochoa accreditation of excellence SEV-2013-0323, the CYTED 2011 project 712RT0449, and the Basque Government Consolidated Research Group Grant IT649-13 on "Mathematical Modeling, Simulation, and Industrial Applications (M2SI)". This research was partially supported by Center for Numerical Porous Media at KAUST.

References

[1] J.A. Cottrell, T.J.R. Hughes, Y. Bazilevs, Isogeometric Analysis: Toward Unification of CAD and FEA, John Wiley and Sons, 2009.
[2] A. El maliki, M. Fortin, N. Tardieu, A. Fortin, Iterative solvers for 3D linear and nonlinear elasticity problems: displacement and mixed formulations, Internat. J. Numer. Methods Engrg. 83 (2010) 1780–1802.
[3] R. Hiptmair, Multigrid method for Maxwell's equations, SIAM J. Numer. Anal. 36 (1) (1998) 204–225.
[4] D.N. Arnold, R.S. Falk, R. Winther, Multigrid in H(div) and H(curl), Numer. Math. 85 (2) (2000) 197–217.
[5] Z. Bazilevs, C. Michler, V.M. Calo, T.J.R. Hughes, Isogeometric variational multiscale modeling of wall-bounded turbulent flows with weakly enforced boundary conditions on unstretched meshes, Comput. Methods Appl. Mech. Engrg. 13–16 (2010) 780–790.
[6] V.M. Calo, H. Gómez, Y. Bazilevs, G.P. Johnson, T.J.R. Hughes, Simulation of engineering applications using isogeometric analysis, in: Proceedings of Tera Grid, 2008.
[7] N. Collier, L. Dalcin, D. Pardo, V.M. Calo, The cost of continuity: performance of iterative solvers on isogeometric finite elements, SIAM J. Sci. Comput. 35 (2) (2013) A767–A784.
[8] A. Buffa, H. Harbrecht, A. Kunoth, G. Sangalli, BPX-preconditioning for isogeometric analysis, Comput. Methods Appl. Mech. Engrg. 265 (2013) 63–70.
[9] L. Gao, V.M. Calo, Preconditioners based on the alternating-direction-implicit algorithm for the 2D steady-state diffusion equation with orthotropic heterogeneous coefficients, J. Comput. Appl. Math. 273 (2015) 274–295.
[10] A. Paszyńska, M. Paszyński, K. Jopek, M. Woźniak, D. Goik, P. Gurgul, H. AbouEisha, M. Moshkov, V.M. Calo, A. Lenerth, D. Nguyen, K. Pingali, Quasi-optimal elimination trees for 2D grids with singularities, Sci. Program. (2014) in press.
[11] D. Goik, K. Jopek, M. Paszyński, A. Lenharth, D. Nguyen, K. Pingali, Graph grammar based multi-thread multi-frontal direct solver with Galois scheduler, Procedia Comput. Sci. 29 (2014) 960–969.

[12] H. AbouEisha, M. Moshkov, V. Calo, M. Paszynski, D. Goik, K. Jopek, Dynamic programming algorithm for generation of optimal elimination trees for multi-frontal direct solver over H-refined grids, Procedia Comput. Sci. 29 (2014) 947–959.

[13] G.H. Golub, C.F. Van Loan, Matrix Computations, third ed., Johns Hopkins University Press, 1996.

[14] B. Irons, A frontal solution program for finite-element analysis, Internat. J. Numer. Methods Engrg. 2 (1970) 5–32.

[15] I.S. Duff, J.K. Reid, The multifrontal solution of indefinite sparse symmetric linear, ACM Trans. Math. Software 9 (3) (1983) 302–325.

[16] P. Geng, T.J. Oden, R.A. van de Geijn, A parallel multifrontal algorithm and its implementation, Comput. Methods Appl. Mech. Engrg. 19 (1997) 289–301.

[17] I.S. Duff, J.K. Reid, The multifrontal solution of unsymmetric sets of linear systems, SIAM J. Sci. Comput. 5 (1984) 633–641.

[18] P.R. Amestoy, A. Guermouche, J.-Y. L'Excellent, S. Pralet, Hybrid scheduling for the parallel solution of linear systems, Comput. Methods Appl. Mech. Engrg. 2 (32) (2001) 136–156.

[19] L. Lin, L.C. Yang, J. Lu, L. Ying, E. Weinan, A fast parallel algorithm for selected inversion of structured sparse matrices wtih application to 2D electronic structure calculations, SIAM J. Sci. Comput. 33 (3) (2011) 1329–1351.

[20] P. Bientinesi, V. Eijkhout, K. Kim, J. Kurtz, R. van de Geijn, Sparse direct factorizations through unassembled hyper-matrices, Comput. Methods Appl. Mech. Engrg. 199 (2010) 430–438.

[21] M. Woźniak, K. Kuźnik, M. Paszyński, V.M. Calo, D. Pardo, Computational cost estimates for parallel shared memory isogeometric multi-frontal solvers, Comput. Math. Appl. 67 (10) (2014) 1864–1883.

[22] M. Paszyński, D. Pardo, A. Paszyńska, Parallel multi-frontal solver for p adaptive finite element modeling of multi-physics computational problems, J. Comput. Sci. 1 (2010) 48–54.

[23] M. Paszyński, D. Pardo, C. Torres-Verdin, L. Demkowicz, V. Calo, A parallel direct solver for self-adaptive hp finite element method, J. Parallel Distrib. Comput. 70 (2010) 270–281.

[24] M. Paszyński, R. Schaefer, Graph grammar driven partial differential eqautions solver, Concurr. Comput.: Pract. Exp. 22 (9) (2010) 1063–1097.

[25] N.O. Collier, D. Pardo, M. Paszyński, L. Dalcín, V.M. Calo, The cost of continuity: a study of the performance of isogeometric finite elements using direct solvers, Comput. Methods Appl. Mech. Engrg. 213–216 (2012) 353–361.

[26] STAMPEDE Linux cluster user guide, Texas Advanced Computing Center, https://www.tacc.utexas.edu/user-services/user-guides/stampede-user-guide, 2014.

[27] N. Collier, L. Dalcin, V.M. Calo, PetIGA: high-performance isogeometric analysis, http://arxiv.org/abs/1305.4452, 2013.

[28] S. Balay, S. Abhyankar, M.F. Adams, J. Brown, P. Brune, K. Buschelman, V. Eijkhout, W.D. Gropp, D. Kaushik, M.G. Knepley, L. Curfman McInnes, K. Rupp, B.F. Smith, H. Zhang, PETSc Web Page, http://www.mcs.anl.gov/petsc, 2014.

[29] S. Balay, S. Abhyankar, M.F. Adams, J. Brown, P. Brune, K. Buschelman, V. Eijkhout, W.D. Gropp, D. Kaushik, M.G. Knepley, L. Curfman McInnes, K. Rupp, B.F. Smith, H. Zhang, PETSc User Manual, Argonne National Laboratory ANL-95/11—Revision 3.4, 2013.

[30] S. Balay, W.D. Gropp, L. Curfman McInnes, B.F. Smith, Efficient management of parallelism in object oriented numerical software libraries, in: E. Arge, A.M. Bruaset, H.P. Langtangen (Eds.), Modern Software Tools in Scientific Computing, Birkhäuser Press, 1997.

[31] P.R. Amestoy, I.S. Duff, Multifrontal parallel distributed symmetric and unsymmetric solvers, Comput. Methods Appl. Mech. Engrg. 184 (2000) 501–520.

[32] P.R. Amestoy, I.S. Duff, J. Koster, J.Y. L'Excellent, A fully asynchronous multifrontal solver using distributed dynamic scheduling, SIAM J. Matrix Anal. Appl. 1 (23) (2001) 15–41.

[33] L.S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, R.C. Whaley, ScaLAPACK Users' Guide, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1999.

[34] Xiaoye S. Li, An overview of SuperLU: algorithms, implementation, and user interface, TOMS Trans. Math. Software 31 (3) (2005) 302–325.

[35] X.S. Li, J.W. Demmel, J.R. Gilbert, iL. Grigori, M. Shao, I. Yamazaki, SuperLU Users' Guide, Lawrence Berkeley National Laboratory, LBNL-44289, http://crd.lbl.gov/˜xiaoye/SuperLU/, 1999.

[36] P. Hénon, P. Ramet, J. Roman, PaStiX: a high-performance parallel direct solver for sparse symmetric definite systems, Parallel Comput. 28 (2) (2002) 301–321.

[37] M. Paszyński, Minimizing the memory usage by out-of-core multi-frontal direct solver, Comput. Assist. Mech. Eng. Sci. 20 (2013) 15–41.