Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie

Wydział Informatyki, Elektroniki i Telekomunikacji

Katedra Informatyki

Praca doktorska

Maciej Woźniak

Solwery izogeometryczne dla różnych architektur maszyn równoległych bazujące na teorii śladów

Promotor:
dr hab. Maciej Paszyński
prof. nadzwyczajny AGH

Kraków 2017

AGH
University of Science and Technology in Krakow

Faculty of Computer Science, Electronics and Telecommunications

Department of Computer Science

Dissertation for the degree of
Doctor of Philosophy

Maciej Woźniak

Trace theory based isogeometric solvers for different parallel architectures

Supervisor:
Maciej Paszyński, Ph.D.
Associate Professor

Krakow 2017

Acknowledgements

I would like to express my deepest gratitude to Professor Maciej Paszyński for supporting me in my research efforts. All his guidance, assistance and persistence that were key to successful completion of my research concluded with this thesis. This research was supported by the National Science Centre, Poland grants OPUS no. DEC-2012/07/B/ST6/01229 "Adaptive and isogeometric parallel strategies for the efficient, accurate solution of difficult non-stationary problems" and PRELUDIUM no. DEC-2014/15/N/ST6/04662 "Alternating directions parallel solver for isogeometric L2 projection", and by Dean's grants from Faculty of Computer Science, Electronics, and Telecommunication. I would also like to acknowledge prof. Victor Calo for supporting my visits at King Abdullah University of Science and Technology (KAUST), prof. Keshav Pingali and prof. Leszek Demkowicz for supporting my visits at Institue for Computational and Engineering Science (ICES), The University of Texas at Austin.

Dedicated to my parents, wife Kamila, and children.

Streszczenie

W niniejszej pracy zaprezentowano teoretyczne oszacowanie kosztu obliczeniowego dokładnego solwera wielofrontalnego dla izogeometrycznej metody elementów skończonych uruchamianego na maszynach równoległych z pamięcią rozproszoną. Teoretyczne szacowanie zarówno kosztu obliczeniowego jak i kosztu komunikacji solwera dokładnego zostało wykonane dla przypadków 2D oraz 3D, z zachowaniem globalnej ciągłości C^{p-1} dla rozwiązania izogeometrycznego.

Następnie zaprezentowano wersję równoległą algorytmu zmienno-kierunkowego solwera izogeometrycznego L^2 projekcji (ADS) do rozwiązywania problemów niestacjonarnych zmiennych w czasie. Algorytm ten został opisany za pomocą pseudokodu. Zaprezentowane są również teoretyczne szacowania kosztu obliczeniowego oraz komunikacji pojedynczego kroku czasowego dla algorytmu równoległego. Przedstawiona jest analiza całkowania dla izogeometrycznej metody elementów skończonych.

W szczególności pokazano, że dla solwerów izogeometrycznych, dla B-spline-ów wyższych rzędów, znaczącym kosztem przy wykonywaniu sekwencyjnym na CPU jest generacja macierzy frontalnych. Algorytm całkowania zaprezentowany został za pomocą sekwencji podstawowych niepodzielnych zadań oraz relacji zależności pomiędzy nimi. Te podstawowe zadania zdefiniowane są dla poszczególnych kroków całkowania dla zadanych punktów całkowania. Zaprezentowany został sposób przygotowywania zestawów niezależnych zadań, które mogą być wykonywane niezależnie na kracie graficznej. Do tego celu został użyty graf pokazujący zależności pomiędzy poszczególnymi zadaniami algorytmu całkowania.

Teoretyczne szacowania dla solwera wielofrontalnego zostały zweryfikowane poprzez wykonanie szeregu eksperymentów z wykorzystaniem trzech równoległych solwerów wielofrontalnych: MUMPS, PaStiX oraz SuperLU, dostępnych w pakiecie obliczeniowym PETIGA, rozszerzającym PETSc. Teoretyczne szacowania dla algorytmu ADS zostały porównane z wynikami eksperymentów wykonanych dla linuxowym klastrze LONESTAR z Texas Advanced Computing Center przy wykorzystaniu 1728 procesorów. Zaprezentowano również zastosowanie algorytmu ADS do rozwiązania wymagającego problemu przepływu nieliniowego w wysoce niejednorodnych ośrodkach porowatych. Algorytm całkowania został zaimplementowany na kartę graficzną oraz przetestowany na szeregu danych numerycznych. Czas obliczeń algorytmu całkowania równoległego na karcie graficznej został porównany z czasem obliczeń algorytmu całkowania sekwencyjnego na CPU. Wszystkie wyniki zostały również porównane z oszacowaniami teoretycznymi.

Abstract

In this thesis we present a theoretical estimation of the computational costs for isogeometric multi-frontal direct solver executed on parallel distributed memory machines. We estimate theoretically both the computational cost and the communication cost of a direct solver for the two-dimensional (2D) case, and for the three-dimensional (3D) case for the C^{p-1} global continuity of the isogeometric solution.

Later we present a parallel version of the alternating directions isogeometric L^2 projections algorithm (ADS) for solving non-stationary time dependent problems. The algorithm is described in pseudo-code. The theoretical estimations on the computational and communication complexities for a single time step of the parallel algorithm are presented. We analyze the integration for isogeometric finite element method solvers.

In particular, we show that isogeometric solvers with higher order B-splines spend significant amount of time for generation of the element frontal matrices when executed sequentially on CPU. The integration algorithm is represented as a sequence of basic undividable computational tasks and the dependency relation between them is identified. The basic tasks are defined for particular steps of the integration

algorithm, for given integration points. We show how to prepare independent sets of tasks that can be automatically scheduled and executed concurrently in a GPU card. This is done with the help of the graph expressing the dependency between tasks, constructed for the integration algorithm.

The theoretical estimates for multi-frontal solver are verified with numerical experiments performed with three parallel multi-frontal direct solvers: MUMPS, PaStiX and SuperLU, available through PETIGA toolkit built on top of PETSc. The theoretical estimates for ADS are compared with numerical experiments performed on the LONESTAR Linux cluster from Texas Advanced Computing Center, using 1728 processors. We also present the application of the method for numerical solution of the challenging problem of nonlinear flows in highly-heterogeneous porous media. The integration algorithm is implemented on GPU and tested on a sequence of numerical examples. The execution time of the concurrent GPU integration is compared with the sequential integration executed on CPU. All resuls have been compared with the theoretical estimates.

Table of contents

Chapter 1

Introduction

1.1. Motivation

The key motivation for this work is:

1. Computing finite element method (just like with any other mesh-based methods) requires generation of two or three-dimensional computational meshes, along with material functions. Classical methods use different polynomials for basis functions to represent geometry and numerical results. We intend to use the isogeometric finite elements method [3, 12, 13, 16, 17], which allows using same functions (B-splines) both for geometry description and solving the numerical problem. We plan to use isogeometric finite elements method to generate the continuous approximation of non-linear parabolic equations. The isogeometric representation allows performing computations for problems requiring higher continuity of basis functions than traditional finite elements method. Current methods of solving nonlinear parabolic problems are very expensive. We expect that using a parallel version of the Alternating Direction Solver (ADS), a recent direct solver algorithm [43], with explicit Euler scheme, will reduce the computational cost significantly. We plan to develop a parallel version of the alternating direction solver, for simulations of nonlinear parabolic problems and their non-linear extensions, using explicit Euler scheme with isogeometric finite elements method.

2. Based on the initial test on isogeometric algorithms in different parallel implementations, we expect that a parallel version of the code for shared memory architecture might be required. Massively parallel model for shared memory is planned. To derive an efficient parallel implementation, we plan to make a formal analysis of the problem concurrency using graph dependency between tasks, based on the practical application of trace theory [35, 61, 78]. We also plan to perform analysis of concurrency of isogeometric solver using PCAM (Partitioning, Communication, Agglomeration, Mapping) methodology [42, 65] targeting distributed memory parallel machines. Additionally, we would like to target hybrid architecture parallel machines, as a result of cascade of the two above models.

3. We plan to estimate the computational and communication complexities of parallel isogeometric solver algorithm on parallel distributed memory machines. We are going to determine the convergence and scalability of the two and three-dimensional model projection algorithms for different input data experimentally, with the application of the simulations of extraction of oil and gas formations in the ground [4, 55].

1.2. Purpose of this book

1.2.1. Main thesis

The main thesis of this book may be summarized as follows:

It is possible to develop new effective direct solver algorithms for Isogeometric Finite Element Method based on concurrency analysis conducted by trace theory. Developed solvers will have better scalability compared to existing direct solvers.

The goal of this dissertation is to propose a parallel trace theory-based algorithm for the isogeometric L^2 projections. The algorithm will allow performing fast simulations with explicit dynamics. They involve a sequence of the isogeometric L^2 projections to be executed in every time step. Our algorithm will deliver an almost ideal scalability on hybrid memory Linux cluster as a result of merging:

(a) parallel Alternating Directions Solver delivering almost ideal parallel scalability on distributed memory Linux cluster and

(b) parallel trace-theory based integration delivering almost ideal parallel scalability on shared memory Linux node and GPGPU

1.2.2. Structure of this book

This book is devided into six chapters and three appendices.
Chapter 2 contains Introduction to sparse matrix multifrontal solvers (Sections 2.1 and 2.2). This is followed by theoretical computational cost estimates (Section 2.3).

Chapter 3 comprises one of two main theoretical parts of this thesis- a parallel version of algorithm from Appendix A. At the beginning (Section 3.1) we present the parallel algorithm itself. It is followed by complexity analysis (Section 3.2) and theoretical cost estimates.

Chapter 4 contains second main theoretical part of this thesis- a new parallel integration algorithm based on trace theory analysis. Section 4.1 introduces integration algorithm, with some details described in Appendix C. Further different types of basis functions are discussed in subsections: in 4.1.1- linear basis functions, quadratic basis functions in 4.1.2 and higher order basis functions in 4.1.3. Selection of tasks and construction of task graph is discussed in Section 4.2. Finally parallel OpenMP implementation is discussed in Section 4.3.

Chapter 5 shows numerical results of algorithms presented in this thesis. Parallel different multi-frontal solvers for IGA-FEM are included in Section 5.1, both in 2D (5.1.1) and 3D (5.1.4). They are followed by parallel isogeometric L^2 projection with ADS algorithm (Section 5.2). Section 5.3 shows proportions between integration and total solution time for both multi-frontal (chapter 2) and ADS (3) solvers. Later we introduce numerical results for comparison of integration time on both GPU and CPU in Section 5.4 as well as OpenMP integration time in Section 5.5. Finally we show application of previous algorithms for solving non-liear flow in heterogeneous media in Section 5.6.

Chapter 6 contains the evaluation of all the obtained results and outlines the potential for future research.

1.3. State of the art

1.3.1. Isogeometric finite element method

Classical higher-order finite element methods (FEM) using hierarchical basis functions [33, 34] maintain C^0-continuity between particular finite elements. The isogeometric analysis (IGA) has been introduced, where B-splines are used as basis functions, and thus, IGA delivers C^k global continuity even on the interfaces between particular finite elements [30].

The higher continuity obtained at element interfaces allows IGA to attain optimal convergence rates for high polynomial orders of approximation, with a low number of degrees of freedom than classical FEM methods [13]. IGA methods allow obtaining the solution of higher-order partial differential equations (PDE) with elegance.

The IGA method has been applied to deformable shell theory [17], phase field modeling [31, 32], phase separation simulations with either Cahn-Hilliard [49] or Navier-Stokes-Korteweg higher order models [50]. The IGA methods have been also succesfully applied for solution of non-linear problems, such as wind turbine aerodynamics [54], incompressible hyper-elasticity [38], turbulent flow simulations [26], transportation of drugs in arterials [53] or the blood flow simulations [14, 23].

Nevertheless, the price to pay for the usage of higher order IGA methods is the higher computational cost of solvers, since the IGA methods produce denser matrices [24, 28]. This is true for all implementations of direct solvers [39, 40], including MUMPS solver [6], modern implementations for adaptive and higher order methods [47] or graph-grammar based approach [67, 68, 69, 70].

1.3.2. Direct and iterative solvers

There are two most popular methods for solving a linear system of equations:

– direct

– iterative

The direct methods deliver an exact solution of the system of linear equations. The only source of error is the round-off error resulting from execution of floating point operations. The iterative methods provide an approximation of the solution, up to the prescribed accuracy. The computational cost of iterative solvers is usually lower then the one of the direct solvers. However, the iterative solvers have several disadvantages:

– They often involve severe convergence problems. Thus, different solvers and different preconditioners are needed for different application, see e.g. [41] for elasticity simulations, [52] for the propagation of electromagnetic waves, [8] for fluid dynamics simulations, [15, 22, 25, 27, 46] for isogeometric finite element method.

– In addition to the convergence problems, iterative solvers may be slower than direct solvers when we solve a problem with multiple right-hand-sides. This is because once we have LU factorized the problem within the direct solver; each new right-hand-side requires just one forward and backward substitution.

– 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 [2, 48, 64].

– Moreover, direct solvers are the main building block of most iterative solvers, e.g. multi-grid solvers.

Thus, direct solvers become essential in many applications.

There exist several direct methods for the solution of a linear system of equations. The fastest method is LU factorization, also known as Gaussian elimination. State-of-the-art implementations of the LU factorization algorithm for sparse matrices include the frontal [39, 56] and multi-frontal solvers [40, 47]. The latest trends in this area include efficient parallelization techniques (see e.g., [7, 60]). Moreover, application-specific implementations take advantage of the data-structures of the Galerkin method, such as the works of [19, 68, 69, 70, 79].

The system of linear equations generated for IGA simulations is usually solved with either multifrontal direct solvers or iterative solvers, such as the ones available through PETSc library [9, 10, 11]. There are also some modern linear computational cost solvers utilizing the concept of H-matrices, delivering approximate solutions [72, 73]. However they work for eliptic positive definite problems only.

In this dissertation, we focus on the isogeometric L^2 projection, which is a numerical kernel for explicit dynamics simulations, and we show how to obtain fast parallel solver for distributed and hybrid memory machines.

1.3.3. Generation of element matrices for finite element method computations

Before we call any solver, either direct or iterative, we need to generate a system of linear equations. It is done by integrating some basis functions over particular finite elements. The formula for the integral changes with the problem being solved, but the structure of the algorithm is the same.

The integration for higher order basis functions executed sequentially on a CPU is often expensive [33, 34]. In general, for higher order basis functions, there are $\mathcal{O}(p)$ functions per single element in one-dimensional problems (1D), and the integration requires $\mathcal{O}(p^3)$ operations. In two-dimensions (2D) integration requires $\mathcal{O}(p^6)$ operations moreover, in three-dimensions (3D), it requires $\mathcal{O}(p^9)$ operations over a single element.

1.3.4. Alternating directions solver for isogeometric L^2 projections

The alternating direction method was introduced to speed up the solution process of time-dependent problems with Laplace equations solved using the finite difference method [20, 37, 71, 77]. The idea was to introduce an intermediate time step and split the Laplace operator into separate operators, the differentiations along particular axes. The similar method has been recently proposed for the case of the computations of the L^2 projections using the isogeometric finite element method [43]. The idea is to utilize the tensor product structure of the B-spline basis functions and to express the projection matrix (equivalent to the mass matrix) as the multiplication of the matrices with B-splines defined along particular axes. The method is described with all the details in Appendix A. This method can also be generalized to curvilinear geometries, by utilizing the conjugate gradient solver with the direction splitting for the preconditioner [44]. Marcin Łoś has estimated the computational complexity of the sequential alternating direction solver for isogeometric L^2 projections in his Master Thesis. We summarize this result in Appendix B.

1.3.5. Isogeometric L^2 projection for time dependent problems

The isogeometric L^2 projections can be applied for solution of time-dependent problems of the following form:

$$\frac{\partial u}{\partial l} - L(u) = f(x,t) \text{ in } \Omega \times (0,T) \tag{1.1}$$

$$u(x,0)=u_0(x) \text{ in } \Omega \tag{1.2}$$

We transform the time dependent problem into the weak form:

$$\left(v, \frac{\partial u}{\partial t}\right)_\Omega - (v, L(u))_\Omega = (v,f)_\Omega \, \forall v \in V \tag{1.3}$$

where

$$(f_1, f_2)_\Omega = \int_\Omega f_1 f_2 \mathrm{d}x \tag{1.4}$$

Finally we can utilize Forward Euler scheme

$$\frac{\partial u}{\partial t} \approx \frac{u_{t+1} - u_t}{\Delta t} \tag{1.5}$$

$$\left(v, \frac{u_{t+1} - u_t}{\Delta t}\right)_\Omega - (v, L(u_t))_\Omega = (v, f_t)_\Omega \, \forall v \in V \tag{1.6}$$

$$(v, u_{t+1})_\Omega = (v, u_t + \Delta t[L(u_t) + f_t])_\Omega \tag{1.7}$$

The Euler scheme is actually equivalent to execution of the sequence of isogeometric L^2 projections. Thus, the parallel alternating directions method for isogeometric L^2 projections can be applied as a fast solver for explicit dynamics.

1.3.6. Open problems

In this dissertation, we plan to solve the following list of open problems, related to the development of efficient direct solvers for three-dimensional simulations of non-stationary parabolic problems of the form

$$\frac{\partial u(x)}{\partial t} - \nabla \circ (K(x) \nabla u) = h(x) \tag{1.8}$$

as well as their non-linear modifications where parameter$K(x)$ is a non-linear function, defined over regular as well as curvilinear computational domain, using isogeometric finite element method [3, 12, 13, 16, 17]. In other words, we have $L_u = \nabla \circ (k(x) \nabla u)$, however our results can be also applied to other kind of L.

1. There exists the Alternating Direction Solver (ADS) sequential algorithm [45] for solutions of particular time steps of the non-stationary problems of the above form Using the Euler scheme and isogeometric finite element method with linear $\mathcal{O}(N)$ computational cost. However, this algorithm does not have an efficient parallel version so far, targeting distributed memory parallel machine.
2. Difficult problems of non-linear flows in heterogeneous media, like the one presented in [4, 55] having the applications to the simulations of extractions of oil and gas formations in the ground, required solution of system of linear equations with several million unknowns (degrees of freedom), to be solved in thousands of time steps. Execution of such the simulations requires the possibility of the solution of systems of linear equations with $10^7 - 10^9$ unknowns within several seconds - several minutes. It is necessary to develop a fast parallel version of the ADS solver working over parallel machines (shared memory nodes, Linux clusters with distributed memory, with local shared memory nodes, GPU cards or clusters of GPU cards with hybrid memory). Only such efficient parallel solver will enable accurate and cost-effective solutions for such applications.

3. The ADS direct solver algorithm works only in the case when the geometry is regular, the Jacobian is constant, and the resulting projection problem is separable [45]. This is related to the case when the geometry of the computational problem is regular and is equal to the three-dimensional cube $[0,1]^3$ [46]. In a case of solving the computational problem over three-dimensional non-regular domains, with curvilinear shapes, defined employing B-splines or NURBS, the Jacobian is a complicated function, which additionally changes from one-time step to the other. In such a case the ADS algorithm can be applied as a preconditioner for iterative solver [35], to reduce the number of iterations of the solver significantly. The only problem is to interface the parallel ADS solver with several parallel iterative solvers. This can be done through interfacing the ADS solver with PETIGA toolkit build on top of PETSc library [1].
4. Classical direct solvers, such as multifrontal direct solver executed on three-dimensional problems has computational cost $\mathcal{O}(N^2p^3)$ in sequential mode and $\mathcal{O}(N^{4/3}p^2)$ in parallel mode, for the case of Linux cluster with distributed memory [81] and similar computational cost for shared memory GPU cards [79]. These computational costs are too large to be able to apply the multifrontal direct solver for large three-dimensional non-stationary problems like the one considered in this work.
5. The development of an efficient parallel version of the ADS solver is necessary for the accurate, efficient solution of a class of three-dimensional non-stationary computational problems considered in this proposal. In particular, the simulations of non-linear flows in heterogeneous media performed so far by using classical methods requires substantial simplifications and rough approximations of input data [4, 55], due to the huge computational cost of direct methods.
6. The B-Spline basis functions of order p have their support over $(p+1)^d$ finite elements, where d is the spatial dimension. This results in $(p+1)^d$ basis functions over each element. Thus generation of element matrices is expensive and it requires some special parallelization techniques.

The main motivation of this dissertation is to obtain a fast, efficient parallel algorithm for the solution of the non-stationary parabolic equations. In particular, we would like to solve the problem of non-linear flow in heterogeneous media [4, 55] using isogeometric finite element method, a modern numerical technique for solving stationary and non-stationary problems [3, 12, 13, 16, 17]. The problem of non-linear flow is important in many geoengineering problems [4, 55], in particular to those related to reservoir characterization and for location and extraction of oil and gas (including shell gas) bearing formations in the ground.

In particular, we aim to solve the following equation

$$\frac{\partial u(x)}{\partial t} - \nabla \circ (K(x,u,\mu)\nabla u) = h(x) \tag{1.9}$$

where u - pressure, K - permeability, h - forcing, domain $D = [0,1]^3$

$$K(x,u,\mu) = K_q(x)e^{10u} \tag{1.10}$$

$$h(x) = 1 + sin(2\pi x_1)sin(2\pi x_2)sin(2\pi x_3) \tag{1.11}$$

where K_q is in range $[1,10]^3$. The values of K_q are presented in Figure 1.1. These values have been generated according to the real data obtained from [4].

We solve the above problem by using Euler scheme in the weak form,

$$(u_{t+1}, v)_{L^2} = (u_t + \Delta t[\nabla \circ (K_q(x)e^{10u}\nabla u_t) + h(x)], v)_{L^2} \forall v \in V \tag{1.12}$$

where V is a space generated by three-dimensional B-spline basis functions. K_q and h are non-uniform in space but constant in time. Initial condition u_0 is three-dimensional ball with maximum value = 2 at point $(0.5, 0.5, 0.5)$, radius=0.33. The time step size is $\Delta t = 10^{-6}$ (in the dimensionless formulation, to satisfy the Courant–Friedrichs–Lewy (CFL) condition of stability).

We would like to develop a fast, efficient parallel alternative direction solver for fast solution of this computational problem.

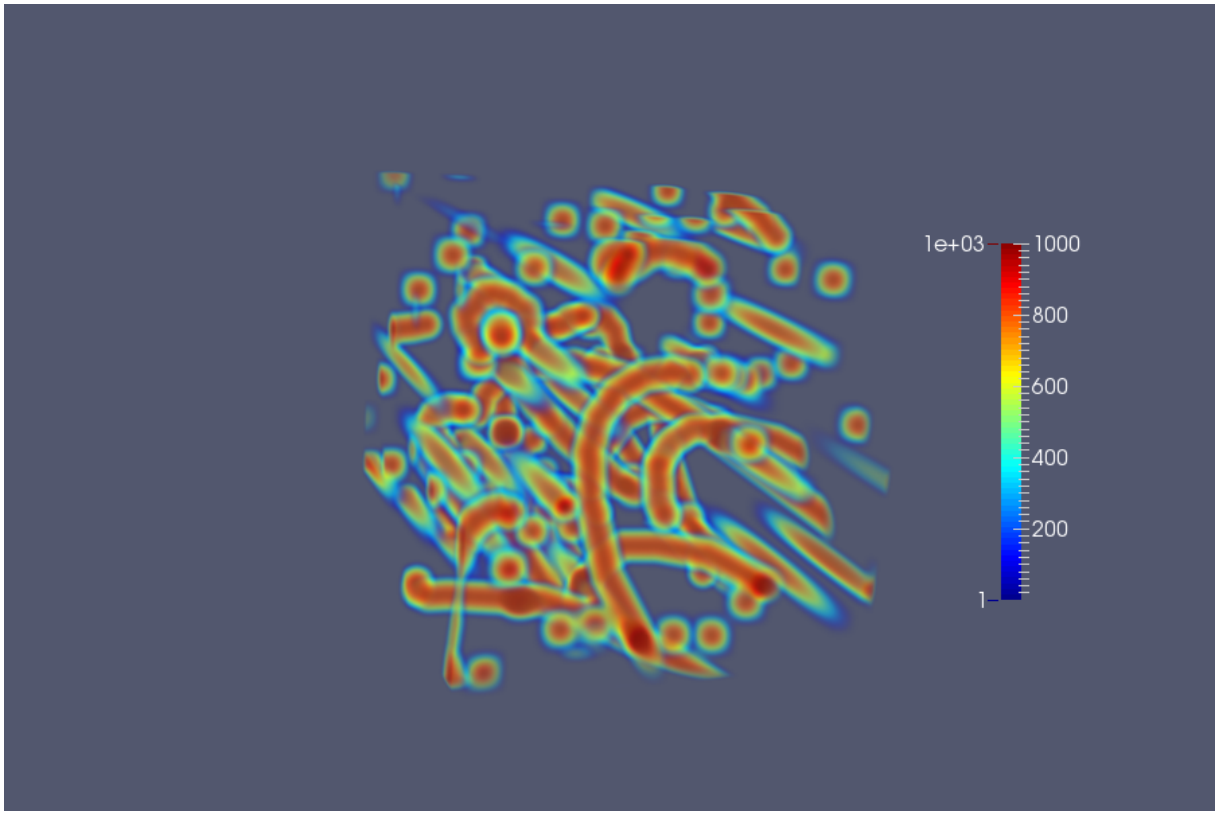


Figure 1.1: Distribution of values of K_q material data over the computational domain.

1.3.7. Main scientific results

The main scientific results of the dissertation are the following:

1. We propose an efficient parallel algorithm of the isogeometric L^2 projection for distributed memory machines, using the trace theory approach.
2. We estimate the computational and communication complexities of state of the art parallel multi-frontal solvers executed over the distributed memory parallel machines
3. We estimate the computational and communication complexities of isogeometric L^2 projections executed over the distributed memory parallel machines
4. We propose efficient algorithm of integration for isogeometric finite element method with B-splines and NURBS for shared memory machines, using the trace theory approach
5. We provide an efficient implementation of the parallel isogeometric L^2 projection for distributed memory parallel machines
6. We provide an efficient implementation of integration for isogeometric finite element method for shared-memory parallel machines

Chapter 2

Computational complexities of classical parallel multi-frontal solver for distributed memory cluster

The main goal of this chapter is to present theoretical estimates of computational complexity for an isogeometric multifrontal direct solver for distributed memory architecture. We derive estimates for the complexity of the number of floating point operations (FLOPS) required to solve a system of linear equations using a direct multifrontal solver on distributed memory parallel machine for two-dimensional and three-dimensional problems.

2.1. Schur Complement

At the very beginning, we analyse cost of Shur Complement operation, since it is the main part of multifrontal solver. By noticing, that Schur Complement is equivalent to partial forward elimination we can estimate FLOPS.

We have a matrix M with q rows to be eliminated (we call these rows "fully assembled rows") from the matrix of size $(q+r) \times (q+r)$. We call the remaining r rows the "non-fully assembled rows".

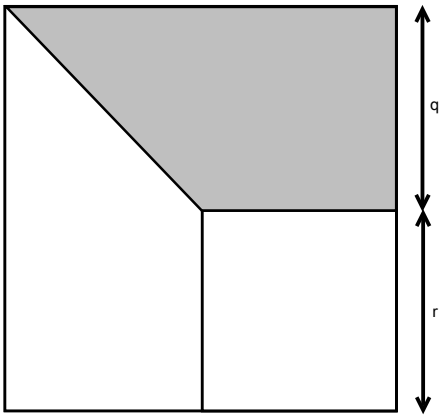


Figure 2.1: Visual explanation of q and r

To perform such elimination we have to eliminate q rows from a matrix with size $(q+r)$, so we have to:

- eliminate of the first row- $(q+r)^2$ subtractions
- eliminate of the second row- $(q+r-1)^2$ subtractions
- ...
- eliminate of the last row- $(r+1)^2$ subtractions

By subtractions we mean the following three things. First, we divide the row by diagonal entry. Next, we subtract the eliminated row from it multiplied by the value from the column below the diagonal of the eliminated row. The exact number of floating-point operations in equation 2.1 involves $3(m+r)^2$ operations instead of $(m+r)^2$ operations since for each entry we perform a multiplication, a division, and subtraction. Then the total number of operations required for partial elimination $S(q,r)$ is:

$$S(q,r) = \sum_{m=1}^{q} 3(m+r)^2 = 3\frac{((r+1)^2+(r+q)^2)q}{2} = \mathcal{O}(q^3+q^2r+qr^2) \tag{2.1}$$

Since we focus on parallel distributed memory machine we assume, that we can use one core per processor. With this assumption in our estimate we can use term "cores" instead of "processors". We also assume that we have as many processors as row subtractions, thus we can perform all subtractions in parallel and reduce the sequential cost $S(q,r)$ to concurrent cost $C(q,r)$ like

$$C(q,r) = \sum_{m=1}^{q} 3(m+r) = 3\frac{((r+1)+(r+q))q}{2} = \mathcal{O}\left(q^2+qr\right) \tag{2.2}$$

2.2. The multi-frontal solver

In order to estimate the computational cost of multi-frontal solver, let us notice that in isogeometric finite element method, B-spline basis functions spread over multiple elements. Let us focus on quadratic B-splines, in 2D, as presented in Figure 2.2.

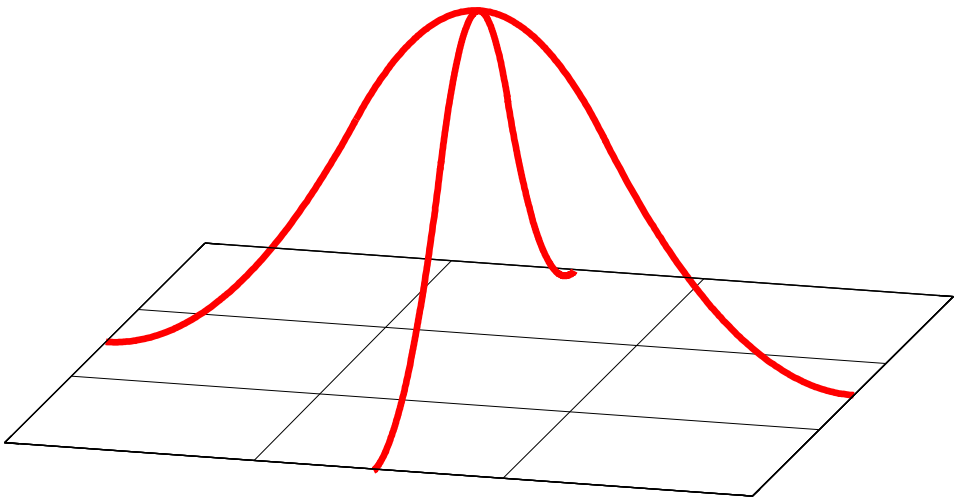


Figure 2.2: Two-dimensional cubic B-Splines spead over $(p+1)^2 = 3^2 = 9$ elements.

Multi-frontal solver merges matrices related to patches of elements. Rows and columns in these matrices are related to B-splines. A single row can be eliminated , if its B-spline is fully assembled. In other words, a single B-spline can be eliminated, if all elements on which it has support, have been processed (merged), and the corresponding matrices have been merged. Let decide that each computational domain contains a cluster of N_p elements. For IGA each patch is a set of p elements in each direction. To simplify let's assume, that the number of clusters in computational domain is $(2^s)^d$, where $s \in \mathbb{Z}$ and d is the spatial dimension of the problem. Even without satisfying this assumption, the final limiting result that we derive here is still true.

The idea of multifrontal solver is to eliminate unknowns related to interior nodes of each cluster, then join each 2^d clusters into one to produce $(2^{s-1})^d$ new clusters and continue with the elimination-join procedure until the last 2^d clusters are joint into one.

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.

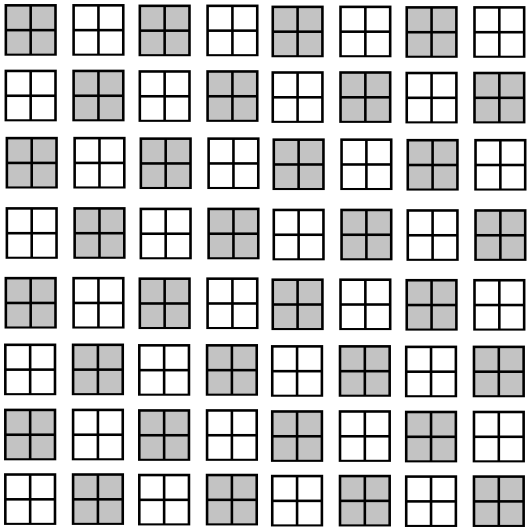
In the parallel setup, the clusters are created on different sub-domains, and the merging process mentioned in step 4 requires inter-processor communications.

An example of the algoritm is ilustrated in Figure 2.3 for the two-dimensional case with C^1 quadratic B-splines, $(2^s)^d$ 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 depicted in Figure 2.3a, we define $N_p(0) = (2^3)^2 = 64$ clusters of elements. Nothing is eliminated in this step.
- In the second step presented in Figure 2.3b we merge sets of four clusters from previous step to create $N_p(1) = (2^2)^2 = 16$ clusters of 16 elements. We eliminate four basis functions from the interior (noted with gray color).
- In the third step displayed in Figure 2.3c we join sets of four clusters from the previous step to create $N_p(2) = (2^1)^2 = 4$ clusters. Next we eliminate 20 basis functions from the interior (noted in Figure 2.3c by dark gray color), and we are left with an interface problem.
- We solve the dense interface problem.

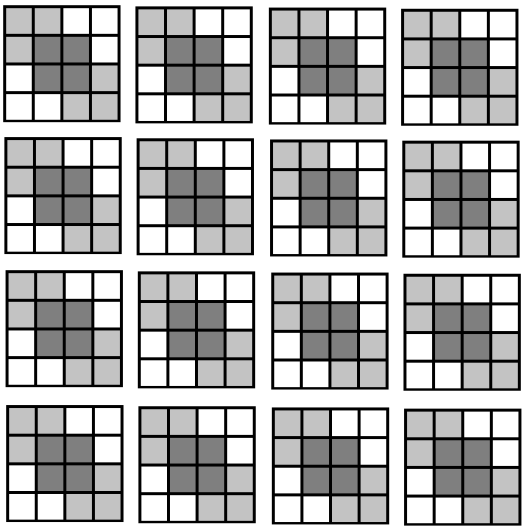
Figure 2.4 ilustrates the algorithm for the three-dimensional case with quadratic B-Splines, $(2^s)^d = (2^2)^3 = 64$ clusters, each one with $p^d = 2^3 = 8$ elements.

- In the first step depicted in Figure 2.4a we have $N_p(0) = 4 \times 4 \times 4 = 64$ patches of $2 \times 2 \times 2 = 8$ elements. We do not eliminate anything.
- In the second step presented in Figure 2.4b we have $N_p(1) - 2 \times 2 \times 2 = 8$ patches of $4 \times 4 \times 4 = 64$ elelemnts obtained from merging 8 patches. We can eliminate $2 \times 2 \times 2 = 8$ basis functions from interior
- Last step concerns one big patch and elimination of remaining 3D cross shape unknowns, plus the boundary. See Figure 2.4c



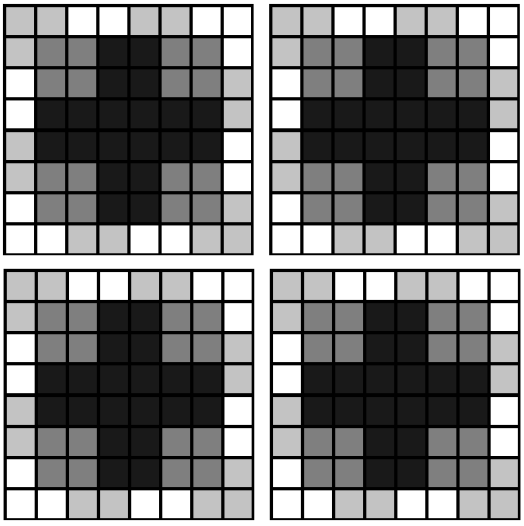
(a)

White and light gray colors denote different clusters of four elements. We have defined here $N_p(0) = (2^3)^2 = 64$. Nothing is eliminated in this step.



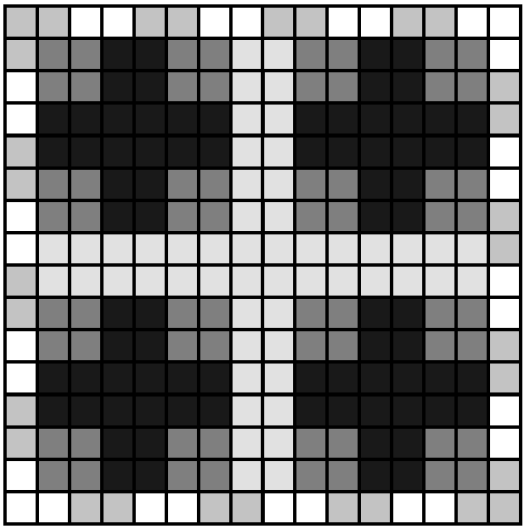
(b)

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. We have created here $N_p(1) = (2^2)^2 = 16$ clusters. We have eliminated 4 basis functions from the interior (noted with grey color).



(c)

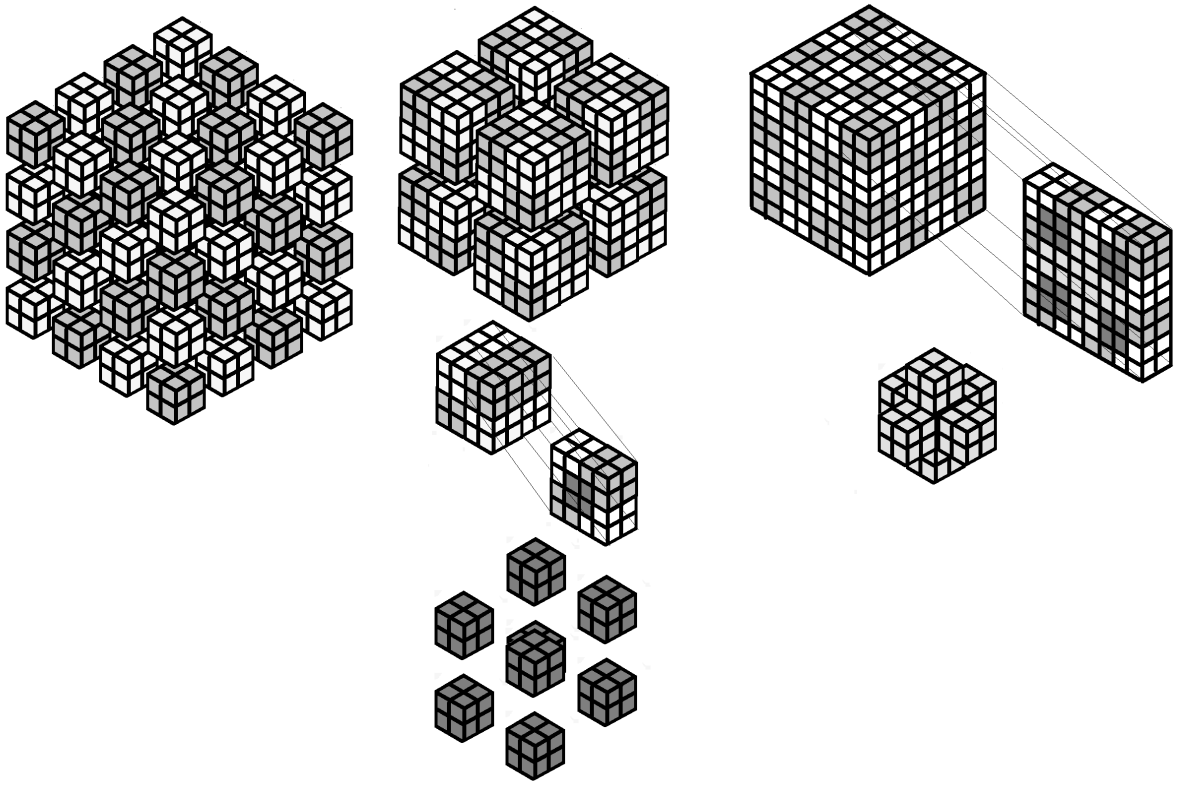
Merging of four clusters, into new clusters of 64 elements. We have created $N_p(2) = (2^1)^2 = 4$ clusters here. Middle-gray color denotes elements whose B-splines have been already eliminated. Dark gray color denotes elements whose B-splines are eliminated at this step.



(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.

Figure 2.3: The scheme of the multi-frontal solver algorithm execution over a two-dimensional grid for quadratic B-splines. Each element contains the entire support of one B-spline with its maximum value attained at its center.



(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.

Figure 2.4: The scheme of the multifrontal solver algorithm execution over a three-dimensional grid for quadratic B-splines. Each element contains the entire support of one B-spline with its maximum value attained at its center.

As shown in [28], the number of FLOPs required by sequential version of the shown algorithm can be expressed as:

$$\sum_{i=0}^{s-1} N_p(i) \cdot S(i) \tag{2.3}$$

where $S(i)$ is the cost of Shur compelement at i-th step, and $s = \log_2(N^{1/d})$, and $N_p(i)$ is number of patches in step i. Following notation of section 2.1 on the Shur complement, we define the number of interior unknowns at the i-th step as $q = q(i)$ and the number of interactiong unknowns at the i-th step as $r = r(i)$.

To estimate the computational and communication costs for the parallel distributed memory version of the multifrontal solver, we assume that we have enough processors available on our Linux cluster and that we have enough memory available on each node. For the estimation of the communication cost we notice that the amount of data exchanged during the communication phase is dominated by the size of the Schur complement matrices exchanged between processors.

Under the given assumptions, the computational cost of parallel solver can be estimated as:

$$\sum_{i=0}^{s-1} C(i) t_{comp} \tag{2.4}$$

where t_{comp} denotes the time of performing a single FLOP operation, $C(i)$ is the cost (in terms of FLOPs) required to perform concurrent Schur complement computations at the i-th step, with concurrent row subtractions executed simultaneously for all the matrices from the current i-th step. The number of patches $N_p(i)$ is not present in 2.4 since we assume that all the patches can be processed in parallel (that we have enough processors available).

The communication cost can be estimated as

$$\sum_{i=0}^{s-1} (t_{init} + C(i) t_{comm}) \tag{2.5}$$

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 equal to the number of entries in the matrix, which is equal to $C(i)$. The number of patches $N_p(i)$ is not present in 2.5 since we assume that we can perform all the communications over one level at the same time (that we have enough communication channels).

Now we can estimate the computational and communication complexities for both sequential and parallel version in the following way:

	$q(0)$	$r(0)$	$q(i)$, $i>0$	$r(i)$, $i>0$	$N_p(i)$	$S_{sequential}(i)$	$S_{parallel}(i)$	$C(i)$
$2D$-IGA C^{p-1}	$\mathcal{O}(1)$	$\mathcal{O}(p^2)$	$\mathcal{O}(2^i p^2)$	$\mathcal{O}(2^i p^2)$	$\mathcal{O}((2^{s-i})^2)$	$\mathcal{O}(2^{3i} p^6)$	$\mathcal{O}(2^{2i} p^4)$	$r(i)^2$
3D-IGA C^{p-1}	$\mathcal{O}(1)$	$\mathcal{O}(p^3)$	$\mathcal{O}(2^{2i} p^3)$	$\mathcal{O}(2^{2i} p^3)$	$\mathcal{O}((2^{s-i})^3)$	$\mathcal{O}(2^{6i} p^9)$	$\mathcal{O}(2^{4i} p^6)$	$r(i)^2$

Table 2.1: Number of interior and interacting unknowns at each step of the multi-frontal solver.

- $q(0)$ stands for number of interior unknowns in the first smallest patch of elements. In 2D these are 2×2 patches, in 3D these are $2 \times 2 \times 2$ patches.
- $r(0)$ defines the number of interacting unknowns in the first smallest patch.
- $q(i)$, $i > 0$ is the number of interior unknowns in the following steps, after merging 4 patches from previous level.
- $r(i)$, $i > 0$ is the number of interface unknowns (B-spline basis functions whose support is not contained inside the patch).
- $N_p(i)$ stands for the number of patches at step i.

In general in 2D we have $\mathcal{O}(2^i p^2)$ unknowns on the interface and interior, and in 3D we have $\mathcal{O}(2^{2i} p^3)$. For the parallel version, we distribute at the i-th step all $N_p(i)$ patches over N_{proc} processors, and if $N_{proc} > N_p(i)$, the cost $S(i)$ over each patch will be further subdivided among the available processors per patch. Now refering to table 2.1 and equation 2.1 we have $q = q(i)$ and $r = r(i)$ and the cost of elimination of interior unknowns is:

$$S(i) = \mathcal{O}(\frac{q(i)^3}{\min\{N_{proc}(i), q(i)\}} + \frac{q(i) r(i)^2}{\min\{N_{proc}(i), r(i)\}}), \text{ where } N_{proc}(i) = \max\{1, \frac{N_{proc}}{N_p(i)}\} \tag{2.6}$$

Notice, that for sufficiently large number of processors N_{proc}, the number of FLOPS reduces to:

$$S(i) = \mathcal{O}(q(i)^2 + q(i)r(i)) \tag{2.7}$$

while for serial implementation the number of FLOPS remains:

$$S(i) = \mathcal{O}(q(i)^3 + q(i)r(i)^2) \tag{2.8}$$

2.3. Summary of the estimates

– 2D IGA:

$$\begin{aligned}
\text{FLOPs} \quad & = \sum_{i=0}^{s-1} N_p(i) \cdot S(i) = N_p(0) \cdot S(0) + \sum_{i=1}^{s-1} N_p(i) \cdot S(i) \\
& = 2^{2s}p^4 + \sum_{i=1}^{s-1} 2^{2(s-i)}2^{3i}p^6 = \mathcal{O}(2^{2s}p^4 + 2^{3s}p^6) \\
& = \mathcal{O}(N_p^3 p^6) = \mathcal{O}(N^{1.5}p^3) \\
\text{Parallel cost} \quad & = \sum_{i=0}^{s-1} C(i)_{comp} + \sum_{i=0}^{s-1} (t_{init} + C(i)t_{comm}) \\
& = r(0)^2 t_{comp} + \sum_{i=1}^{s-1} C(i)_{comp} + t_{init} + r(0)^2 t_{comm} + \sum_{i=1}^{s-1} (t_{init} + C(i)t_{comm}) \\
& = t_{init} + p^4 t_{comp} + \sum_{i=1}^{s-1} 2^{2i}p^4 t_{comp} + p^4 t_{comm} + \sum_{i=1}^{s-1} \left(t_{init} + 2^{2i}p^4 t_{comm}\right) \\
& = \mathcal{O}(p^4 t_{comp} + 2^{2s}p^4 t_{comp} + st_{init} + p^4 t_{comm} + 2^{2s}p^4 t_{comm}) \\
& = \mathcal{O}(Np^2 t_{comp} + \log(N^{0.5})t_{init} + Np^2 t_{comm})
\end{aligned} \tag{2.9}$$

– 3D IGA:

$$\begin{aligned}
\text{FLOPs} \quad & = \sum_{i=0}^{s-1} N_p(i) \cdot S(i) = N_p(0) \cdot S(0) + \sum_{i=1}^{s-1} N_p(i) \cdot S(i) \\
& = 2^{3s}p^6 + \sum_{i=1}^{s-1} 2^{3(s-i)}2^{6i}p^9 = \mathcal{O}(2^{3s}p^6 + 2^{6s}p^9) \\
& = \mathcal{O}(N_p^3 p^6 + N_p^6 p^9) = \mathcal{O}(N^2 p^3) \\
\text{Parallel cost} \quad & = \sum_{i=0}^{s-1} C(i)_{comp} + \sum_{i=0}^{s-1} (t_{init} + C(i)t_{comm}) \\
& = r(0)^2 t_{comp} + \sum_{i=1}^{s-1} C(i)_{comp} + t_{init} + r(0)^2 t_{comm} + \sum_{i=1}^{s-1} (t_{init} + C(i)t_{comm}) \\
& = t_{init} + p^6 t_{comp} + \sum_{i=1}^{s-1} 2^{4i}p^6 t_{comp} + p^6 t_{comm} + \sum_{i=1}^{s-1} \left(t_{init} + 2^{4i}p^6 t_{comm}\right) \\
& = \mathcal{O}(p^6 t_{comp} + 2^{4s}p^6 t_{comp} + st_{init} + 2^{4s}p^6 t_{comm}) \\
& = \mathcal{O}(N^{4/3}p^2 t_{comp} + \log(N^{1/3})t_{init} + N^{4/3}p^2 t_{comm})
\end{aligned} \tag{2.10}$$

From our estimates, it implies that both computation and communication complexities are of the order of $\mathcal{O}(Np^2)$ (2D IGA-FEM) and $\mathcal{O}(N^{4/3}p^2)$ (3D IGA-FEM). These are relatively high computation and communication complexities, and thus there is a need to design and efficient parallel for isogeometric L2 projections, that can be used in parallel explicit dynamics simulations of nonlinear flow in heterogenous media. One possibility is to develop a parallel version of L^2 projections algorithm described in Appendix A. This is a subject of next chapter.

Chapter 3

Parallel alternating direction algorithm for isogeometric L^2 projections

In this section we perform a classical Partitioning, Communication, Agglomeration, and Mapping (PCAM) [42] analysis of the computational and communication complexities of the parallel alternating direction algorithm for isogeometric L^2 projections.

3.1. Parallel Isogeometric L^2 Projection Algorithm

The sequential version of ADS is described in Appendix A. We propose a parallel version of the algorithm [80], targeting distributed memory Linux cluster parallel machines.

The parallel variant of the isogeometric L^2 projection algorithm generates data distributed in a uniform way over a three-dimensional cube of processors, as depicted in Figure 3.1. There are three steps of the algorithm where the data are gathered into OYZ, OXZ and OXY faces, respectively. The local 1D banded problems are solved there, using the LAPACK library. The data are scattered into a cube of processors, to proceed with the next step. The algorithm can be summarized as shown in Figure 3.1. The algorithm performs isogeometric L^2 projection over a cube domain with tensor product of $N_x \times N_y \times N_z$ 1D B-spline basis functions along x, y and z directions.

0 Integration

1a Gather within every row of processors into OYZ face
1b Solve $N_y N_z$ 1D problems with N_x right hand sides
1c Scatter results onto cube of processors
1d Reorder right hand sides

2a Gather within every row of processors into OXZ face
2b Solve $N_x N_z$ 1D problem with N_y right hand sides
2c Scatter results onto cube of processors
2d Reorder right hand sides

3a Gather in every row of processors into OXY face

3b Solve $N_x N_y$ 1D problem with N_z right hand sides

3c Scatter results onto cube of processors

3d Reorder right hand sides

This algorithm is dedicated for distributed memory Linux cluster nodes. Later in Chapter 4, we will show that by using the trace theory approach we can improve the scalability of the algorithm by one or two orders of magnitude when switching to the hybrid memory parallel machines (namely the distributed memory Linux cluster with all the nodes equipped with GPGPU).

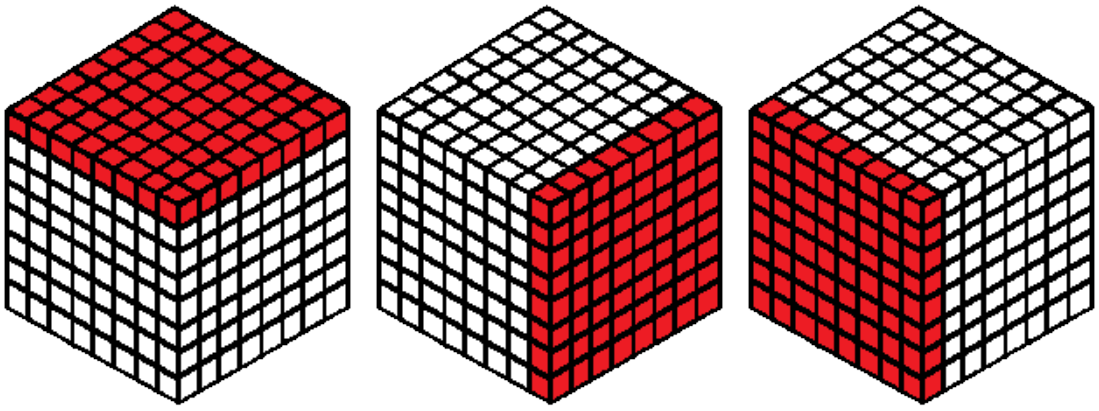


Figure 3.1: Gathering and scattering data into three faces of the three-dimensional cube of processors

3.2. Complexity analysis

Complexity analysis is similar to the one presented in Appendix B.

3.2.1. Integration over one element

Every element is approximated by a set of polynomials in each direction where p is the order, and there are $p+1$ B-splines over the element. We denote p_x as the degree in x direction and p_y and p_z as degrees in other directions.

The integration of the right hand side requires using Gaussian quadrature with $(p_x+1)(p_y+1)(p_z+1)$ points. The integral over each element is:

$$\sum_{m=1}^{(p_x+1)(p_y+1)(p_z+1)} w_m B_x^i(x_m) B_y^j(y_m) B_z^k(z_m) f(x_m, y_m, z_m) dxdydz \tag{3.1}$$

where w_m denotes the Gaussian quadrature weights, B_x^i, B_y^j, B_z^k denote the B-spline basis functions in x, y, and z directions respectively, computed at x_m, y_m, z_m Gaussian quadrature points, and we have $i = 1, \ldots, p_x+1$, $j = 1, \ldots, p_y+1$ and $k = 1, \ldots, p_z+1$ entries to compute. Assume that for $d = 1, \ldots, (p_x+1)(p_y+1)(p_z+1)$ counting value at given point for given element and function f costs $\Phi^f((p_x+1)^2(p_y+1)^2(p_z+1)^2)$ arithmetic operations where Φ^f is the function depending on f.
The formula for Φ^f depends on the form of f. If f is given by a prescribed formula, then cost of computing a value of f is constant and Φ^f is constant. Otherwise when f is given by a combination of B-splines

$$f = \sum_{o=1}^{p_x+1} \sum_{q=1}^{p_y+1} \sum_{r=1}^{p_z+1} B_x^o B_y^q B_z^r f_{oqr} \tag{3.2}$$

then

$$\Phi^f(x_m) = (p_x + 1)(p_y + 1)(p_z + 1) \tag{3.3}$$

and total cost will be

$$(p_x + 1)^3(p_y + 1)^3(p_z + 1)^3 \tag{3.4}$$

In the following part of the paper we assume that f is described by a given formula, and so the cost of computation a value of f at given point is constant.

3.2.2. Integration over all elements

Since we have a mesh of $N_x \times N_y \times N_z$ elements (where N_x, N_y, N_z denotes the number of elements in the x, y and z direction, respectively) the total cost of integration will be

$$(p_x + 1)^2(p_y + 1)^2(p_z + 1)^2 N_x N_y N_z \Phi^f \tag{3.5}$$

We can do every integration with zero communication cost. When we have cuboid of $c_x c_y c_z$ cores it can be done in:

$$\frac{(p_x + 1)^2(p_y + 1)^2(p_z + 1)^2 N_x N_y N_z \Phi^f}{c_x c_y c_z} \tag{3.6}$$

with computational complexity of

$$\mathcal{O}(\frac{p_x^2 p_y^2 p_z^2 N_x N_y N_z}{c_x c_y c_z}) \tag{3.7}$$

3.2.3. Solve

In each step of the algorithm we LU factorize a banded matrix resulting from one dimensional B-spline basis function of order p. We do it on a face of the three-dimensional cuboid of processors. Let N be the number of elements in one direction. Then, the banded matrix M^N of size N with $2p + 1$ diagonal blocks can be LU factorized in $\mathcal{O}(p^2 N)$ steps.
When solving problem in the x direction we have to LU factorize matrix M^{N_x} of size N_x with $2p_x + 1$ diagonal blocks and we have $\frac{N_y}{c_y} \times \frac{N_z}{c_z}$ right hand sides, each one of size N_x. The communication cost is zero since we have $c_y \times c_z$ CPUs solving sequentially at the same time. Solving in x direction using Thomas algorithm over each of these processors results in a computational complexity

$$\mathcal{O}\left(N_x p_x^2 \frac{N_y}{c_y} \frac{N_z}{c_z}\right) \tag{3.8}$$

The solution complexity over y and z directions can be estimated in analogous way as

$$\mathcal{O}\left(N_y p_y^2 \frac{N_x}{c_x} \frac{N_z}{c_z}\right) \tag{3.9}$$

and

$$\mathcal{O}\left(N_z p_z^2 \frac{N_x}{c_x} \frac{N_y}{c_y}\right) \tag{3.10}$$

this results in computational complexity

$$\mathcal{O}\left(\frac{(p_x^2 c_x + p_y^2 c_y + p_z^2 c_z)(N_x N_y N_z)}{c_x c_y c_z}\right) \tag{3.11}$$

3.2.4. Gathering data

While collecting data in x direction we need to collect information from $c_x c_y c_z - c_y c_z$ CPUs into $c_y c_z$ CPUs. Each processor has $\frac{N_x}{c_x}\frac{N_y}{c_y}\frac{N_z}{c_z}$ data. We apply a torus communication structure available on a linux cluster. This implies linear communication complexity in each row of processors in each direction and gives us communication complexity of:

$$\mathcal{O}\left(N_x \frac{N_y}{c_y}\frac{N_z}{c_z}\right) \tag{3.12}$$

Additionally, the gathering data along y and z directions results in the communication complexity of:

$$\mathcal{O}\left(N_y \frac{N_x}{c_x}\frac{N_z}{c_z}\right) \tag{3.13}$$

and

$$\mathcal{O}\left(N_z \frac{N_x}{c_x}\frac{N_y}{c_y}\right) \tag{3.14}$$

Summing, the total communication complexity of gathering data is equal to

$$\mathcal{O}\left(\frac{(c_x+c_y+c_z)N_x N_y N_z}{c_x c_y c_z}\right) \tag{3.15}$$

3.2.5. Reorder data

After processing data in the x-direction we need to perform the reorder of data over each CPU before processing data along the y direction. Similar reordering applies after processing data in the y-direction and before processing data in the z-direction. The computational complexity of each of the two reorders, executed over each processor is equal to

$$\mathcal{O}\left(\frac{N_x N_y N_z}{c_x c_y c_z}\right) \tag{3.16}$$

3.2.6. Scattering data

Scatter is just an inverse of the gather, and its communication complexity is the same as the cost of the gather operation

$$\mathcal{O}\left(\frac{(c_x+c_y+c_z)(N_x N_y N_z)}{c_x c_y c_z}\right) \tag{3.17}$$

3.2.7. Total complexity

From the discussion above, we conclude that we can construct isogeometric projection solver with the total cost

$$\begin{aligned}\left(\frac{p_x^2 p_y^2 p_z^2 N_x N_y N_z}{c_x c_y c_z}\right) t_{comp} + \left(\frac{(p_x^2 c_x + p_y^2 c_y + p_z^2 c_z)(N_x N_y N_z)}{c_x c_y c_z}\right) t_{comp} + \\ + \left(\frac{N_x N_y N_z}{c_x c_y c_z}\right) t_{comp} + \left(\frac{(c_x+c_y+c_z)N_x N_y N_z}{c_x c_y c_z}\right) t_{comm}\end{aligned} \tag{3.18}$$

for arbitrary polynomial orders p_x, p_y, p_z, dimension sizes N_x, N_y, N_z and processors numbers c_x, c_y, c_z, where t_{comp} is the cost of processing a single floating point operation, and t_{comm} is the cost of communicating a single byte.

Assuming

$$N_x = N_y = N_z = N^{1/3},\ p_x = p_y = p_z = p,\ c_x = c_y = c_z = c^{1/3} \tag{3.19}$$

we have the following cost

$$\left(\frac{p^6 N}{c} + \frac{p^2 N}{c^{2/3}} + \frac{p^3 N}{c}\right) t_{comp} + \left(\frac{N}{c^{2/3}}\right) t_{comm} \tag{3.20}$$

which implies the computational complexity

$$\mathcal{O}\left(p^6 \frac{N}{c}\right) \tag{3.21}$$

and communication complexity

$$\mathcal{O}\left(\frac{N}{c^{2/3}}\right) \tag{3.22}$$

The most expensive part of the algorithm is the integration, but the good news is that it is almost perfectly parallelizable on distributed, shared and hybrid memory architectures, as we will show in the next chapters.

Chapter 4

Trace theory based analysis of concurrency of the integration for IGA-FEM

The purpose of this chapter is to present trace theory based analysis of concurrency of the integration, which is the most expensive part of the alternating direction solver for isogeometric L2 projections. We refer to Appendix C for the derivation of the element matrices and right-hand-sides for the isogeometric projection problem. We focus on 2D problem for simplicity, but this result is extendedible to 3D case as well.

4.1. The integration algorithm

We start from the element matrices and right hand sides for the isogeometric L^2 projection problem as defined in Appendix C.

$$\sum_{i=1,\dots,N_x,j=1,\dots,N_y} a_{i,j}(B_{i,j;p}, B_{k,l;p}) = (F, B_{k,l;p}) \; k = 1,\dots,N_x, l = 1,\dots,N_y \tag{4.1}$$

Using Gaussian quadrature, the definition of $B_{k,l;p}(x_1, x_2) = N_{k;p}(x_1)N_{l;p}(x_2)$, and the definition of the scalar product we can see that the integration over the domain can be presented by a weighted summation over Gaussian points.

$$\begin{gathered}\int_\Omega N_{i;p}(x_1)N_{j;p}(x_2)N_{k;p}(x_1)N_{l;p}(x_2)\mathrm{d}x_1\mathrm{d}x_2 = \sum_n w_n N_{i;p}(x_1^n)N_{j;p}(x_2^n)N_{k;p}(x_1^n)N_{l;p}(x_2^n) \\ \forall i,k = 1,\dots,N_x, j,l = 1,\dots,N_y\end{gathered} \tag{4.2}$$

and

$$\begin{gathered}\int_\Omega N_{k;p}(x_1)N_{l;p}(x_2)\mathrm{d}x_1\mathrm{d}x_2 = \sum_n w_n N_{k;p}(x_1^n)N_{l;p}(x_2^n) \\ \forall k = 1,\dots,N_x, l = 1,\dots,N_y\end{gathered} \tag{4.3}$$

for a given p, where (x_n^1, x_n^2) and w_n denotes the Gaussian quadrature integration points and weights. For interfacing with either a direct or an iterative solver, an alternating directions solver, the mesh is partitioned into "elements", as illustrated in Figure 4.1. Next, element frontal matrices are generated by performing the integration over particular elements.

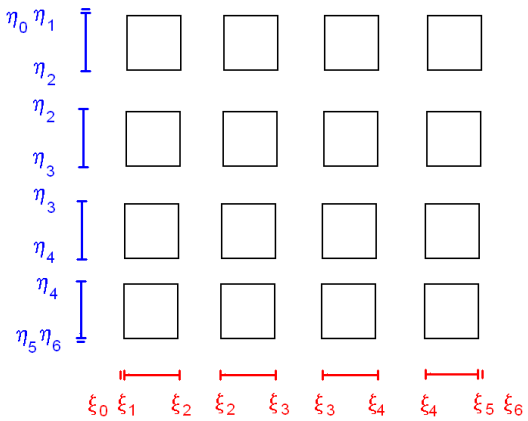


Figure 4.1: Partition of the computational mesh into elements

4.1.1. Linear basis functions

For linear basis functions, we take $2 * 2 = 4$ two-dimensional B-splines, each of which is the tensor product of two one-dimensional B-splines. This is illustrated in Figure 4.2.

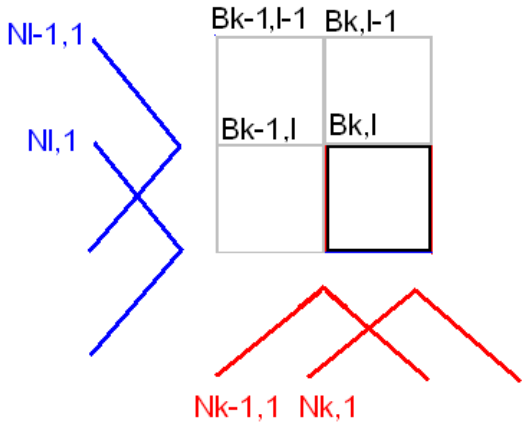


Figure 4.2: Supports of linear B-spline basis functions over a single element

$(B_{k-1,l-1;1}, B_{k-1,l-1;1})$	$(B_{k,l-1;1}, B_{k-1,l-1;1})$	$(B_{k-1,l;1}, B_{k-1,l-1;1})$	$(B_{k,l;1}, B_{k-1,l-1;1})$
$(B_{k-1,l-1;1}, B_{k,l-1;1})$	$(B_{k,l-1;1}, B_{k,l-1;1})$	$(B_{k-1,l;1}, B_{k,l-1;1})$	$(B_{k,l;1}, B_{k,l-1;1})$
$(B_{k-1,l-1;1}, B_{k-1,l;1})$	$(B_{k,l-1;1}, B_{k-1,l;1})$	$(B_{k-1,l;1}, B_{k-1,l;1})$	$(B_{k,l;1}, B_{k-1,l;1})$
$(B_{k-1,l-1;1}, B_{k,l;1})$	$(B_{k,l-1;1}, B_{k,l;1})$	$(B_{k-1,l;1}, B_{k,l;1})$	$(B_{k,l;1}, B_{k,l;1})$

Table 4.1: Frontal matrix with linear basis functions and the corresponding tasks names

A frontal matrix for the case of linear basis functions is illustrated in Table 4.1. The frontal matrix is obtained from integration over a single finite element $E_{k,l}$, where four basis functions have non-zero support, namely $B_{k-1,l-1;1}, B_{k,l-1;1}, , B_{k-1,l;1}, B_{k,l;1}$. Thus, the rows and columns of the frontal matrix correspond to the four basis functions, moreover, its entries contain scalar products of all the combinations of the basis functions. In other words, indices k, $k-1$, l, $l-1$ are used for element $E_{k,l}$ relative numbering of basis functions. The considerations presented here are identical for all elements $E_{k,l}$. Each entry of the

frontal matrix is obtained by summing up the values of the scalar products at Gaussian integration points. The computational tasks of evaluating the value of the scalar product at Gaussian integration point is presented in Table 4.2 and denoted by $t^1_{i,j;k,l}$.

$t^1_{k-1,l-1;k-1,l-1} =$ $(B_{k-1,l-1;1}(x_1,x_2), B_{k-1,l-1;1}(x_1,x_2))$	$t^1_{k,l-1;k-1,l-1} =$ $(B_{k,l-1;1}(x_1,x_2), B_{k-1,l-1;1}(x_1,x_2))$
$t^1_{k-1,l;k-1,l-1} =$ $(B_{k-1,l;1}(x_1,x_2), B_{k-1,l-1;1}(x_1,x_2))$	$t^1_{k,l;k-1,l-1} =$ $(B_{k,l;1}(x_1,x_2), B_{k-1,l-1;1}(x_1,x_2))$
$t^1_{k-1,l-1;k,l-1} =$ $(B_{k-1,l-1;1}(x_1,x_2), B_{k,l-1;1}(x_1,x_2))$	$t^1_{k,l-1;k,l-1} =$ $(B_{k,l-1;1}(x_1,x_2), B_{k,l-1;1}(x_1,x_2))$
$t^1_{k-1,l;k,l-1} =$ $(B_{k-1,l;1}(x_1,x_2), B_{k,l-1;1}(x_1,x_2))$	$t^1_{k,l;k,l-1} =$ $(B_{k,l;1}(x_1,x_2), B_{k,l-1;1}(x_1,x_2))$
$t^1_{k-1,l-1;k-1,l} =$ $(B_{k-1,l-1;1}(x_1,x_2), B_{k-1,l;1}(x_1,x_2))$	$t^1_{k,l-1;k-1,l} =$ $(B_{k,l-1;1}(x_1,x_2), B_{k-1,l;1}(x_1,x_2))$
$t^1_{k-1,l;k-1,l} =$ $(B_{k-1,l;1}, B_{k-1,l;1}(x_1,x_2))$	$t^1_{k,l;k-1,l} =$ $(B_{k,l;1}, B_{k-1,l;1}(x_1,x_2))$
$t^1_{k-1,l-1;k,l} =$ $(B_{k-1,l-1;1}(x_1,x_2), B_{k,l;1}(x_1,x_2))$	$t^1_{k,l-1;k,l} =$ $(B_{k,l-1;1}(x_1,x_2), B_{k,l;1}(x_1,x_2))$
$t^1_{k-1,l;k,l} =$ $(B_{k-1,l;1}(x_1,x_2), B_{k,l;1}(x_1,x_2))$	$t^1_{k,l;k,l} =$ $(B_{k,l;1}(x_1,x_2), B_{k,l;1}(x_1,x_2))$

Table 4.2: Computational tasks responsible for evaluation of the values of scalar products of two-dimensional linear basis functions over element $E_{k,l}$ at Gaussian quadrature points.

In such case, it is necessary to compute $2 * 2 = 4$ linear basis functions at Gaussian integration points, as presented in Table 4.3. Again, in this table, we have named particular tasks by $t^1_{i,j}$.

$t^1_{k,l} = B_{k,l;1}(x_1,x_2) = N_{k;1}(x_1)N_{l;1}(x_2)$
$t^1_{k,l-1} = B_{k,l-1;1}(x_1,x_2) = N_{k;1}(x_1)N_{l;1}(x_2)$
$t^1_{k-1,l} = B_{k-1,l;1}(x_1,x_2) = N_{k-1;1}(x_1)N_{l;1}(x_2)$
$t^1_{k-1,l-1} = B_{k-1,l-1;1}(x_1,x_2) = N_{k-1;1}(x_1)N_{l;1}(x_2)$

Table 4.3: Computational tasks responsible for evaluation of the values of two-dimensional linear basis functions over element $E_{k,l}$ at Gaussian quadrature points.

The computational tasks of evaluating the linear basis functions involve tensor products of $2 + 2 = 4$ one-dimensional linear B-splines. This is represented in Table 4.4, where we have named particular tasks by t^1_i.

$t_k^1 =$ $N_{k;1}(x_1)$	$t_{k-1}^1 =$ $N_{k-1;1}(x_1)$
$t_l^1 =$ $N_{l;1}(x_2)$	$t_{l-1}^1 =$ $N_{l-1;1}(x_2)$

Table 4.4: Computational tasks responsible for evaluation of the values of one dimensional linear basis functions over element $E_{k,l}$ at Gaussian quadrature points.

4.1.2. Quadratic basis functions

For quadratic basis functions, we have $3 * 3 = 9$ tensor products of two one-dimensional B-splines. This is illustrated in Figure 4.3.

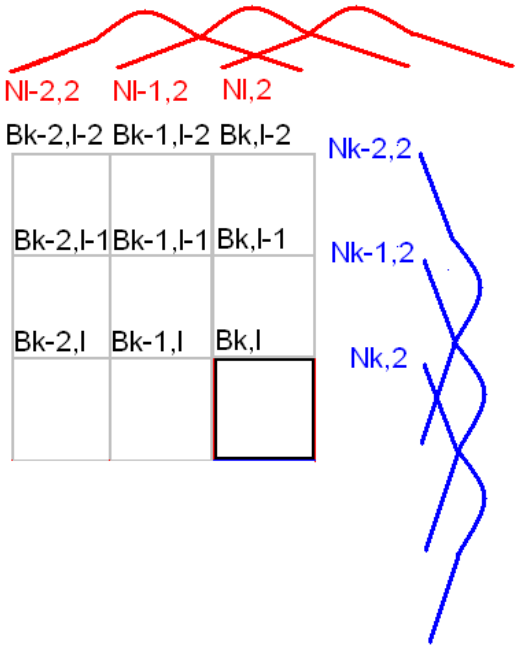


Figure 4.3: Quadratic B-splines over a single element

A frontal matrix for the case of quadratic basis functions is illustrated in Table 4.5. The frontal matrix is obtained from integration over a single finite element $E_{k,l}$, where nine basis functions have non-zero support, namely $B_{k-2,l-2;2}, \ldots, B_{k,l;1}$. Thus, the rows and columns of the frontal matrix correspond to the nine basis functions, and its entries contain scalar products of all the combinations of the basis functions. In other words, indices k, $k-1$, $k-2$ and l, $l-1$, $l-2$ are used for element $E_{k,l}$ relative numbering of basis functions. The considerations presented here are identical for all elements $E_{k,l}$. The polynomial order of approximation $p = 2$ is fixed. Each entry of the frontal matrix is obtained by summing up the values of the scalar products at Gaussian integration points. The computational tasks of computing the value of the scalar product at Gaussian integration point is presented in Table 4.6 and denoted by $t^2_{i,j;k,l}$.

In such case, it is necessary to compute $3 * 3 = 9$ quadratic basis functions at Gaussian quadrature points, presented in Table 4.7, with tasks $t^2_{i,j}$.

The computational tasks of computing the quadratic basis functions involve tensor products of $3+3 = 6$ one-dimensional quadratic B-splines at the points, as presented in Table 4.8. The tasks are named as t_i^2.

$(B_{k-2,l-2;2}, B_{k-2,l-2;2})$	$\dots$	$(B_{k,l;2}, B_{k-2,l-2;2})$
$\dots$	$\dots$	$\dots$
$(B_{k-2,l-2;2}, B_{k,l;2})$	$\dots$	$(B_{k,l;2}, B_{k,l;2})$

Table 4.5: Frontal matrix with quadratic basis functions

$t^2_{k-2,l-2;k-2,l-2} =$ $(B_{k-2,l-2;2}(x_1,x_2), B_{k-2,l-2;2}(x_1,x_2))$	$\dots$	$t^2_{k,l;k-2,l-2} =$ $(B_{k,l;2}(x_1,x_2), B_{k-2,l-2;2}(x_1,x_2))$
$\dots$	$\dots$	$\dots$
$t^2_{k-2,l-2;k,l} =$ $(B_{k-2,l-2;2}(x_1,x_2), B_{k,l;2}(x_1,x_2))$	$\dots$	$t^2_{k,l;k,l} =$ $(B_{k,l;2}(x_1,x_2), B_{k,l;2}(x_1,x_2))$

Table 4.6: Computational tasks responsible for evaluation of the values of scalar products of two-dimensional quadratic basis functions over element $E_{k,l}$ at Gaussian quadrature points.

$t^2_{k,l} = B_{k,l;2}(x_1,x_2)$ $= N_{k;2}(x_1)N_{l;2}(x_2)$	$t^2_{k,l-1} = B_{k,l-1;2}(x_1,x_2)$ $= N_{k;2}(x_1)N_{l-1;2}(x_2)$	$t^2_{k,l-2} = B_{k,l-2;2}(x_1,x_2)$ $= N_{k;2}(x_1)N_{l-2;2}(x_2)$
$t^2_{k-1,l} = B_{k-1,l;2}(x_1,x_2)$ $= N_{k-1;2}(x_1)N_{l;2}(x_2)$	$t^2_{k-1,l-1} = B_{k-1,l-1;2}(x_1,x_2)$ $= N_{k-1;2}(x_1)N_{l-1;2}(x_2)$	$t^2_{k-2,l-1} = B_{k-1,l-2;2}(x_1,x_2)$ $= N_{k-1;2}(x_1)N_{l-2;2}(x_2)$
$t^2_{k-2,l} = B_{k-2,l;2}(x_1,x_2)$ $= N_{k-2;2}(x_1)N_{l;2}(x_2)$	$t^2_{k-2,l-1} = B_{k-2,l-1;2}(x_1,x_2)$ $= N_{k-2;2}(x_1)N_{l-1;2}(x_2)$	$t^2_{k-2,l-2} = B_{k-2,l-2;2}(x_1,x_2)$ $= N_{k-2;2}(x_1)N_{l-2;2}(x_2)$

Table 4.7: Computational tasks responsible for evaluation of the values of two-dimensional quadratic basis functions over element $E_{k,l}$ at Gaussian quadrature points.

$t^2_k = N_{k;2}(x_1)$	$t^2_{k-1} = N_{k-1;2}(x_1)$	$t^2_{k-2} = N_{k-2;2}(x_1)$
$t^2_l = N_{l;2}(x_2)$	$t^2_{l-1} = N_{l-1;2}(x_2)$	$t^2_{l-2} = N_{l-2;2}(x_2)$

Table 4.8: Computational tasks responsible for evaluation of the values of one dimensional quadratic basis functions over element $E_{k,l}$ at Gaussian quadrature points.

4.1.3. Higher order basis functions

The scheme presented in subsection 4.1.1 and 4.1.2 for linear and quadratic basis functions, respectively, can be generalized for arbitrary higher order basis functions.

In particular, over a single element $E_{k,l} = [\xi_K, \xi_{K+1}] \times [\eta_L, \eta_{L+1}]$ there are $(p+1)(p+1)$ basis functions defined as tensor products of one dimensional B-splines of order p

$$\{B_{m,n;p}(x_1,x_2)\}_{m=k-p,\dots,k;n=l-p,\dots,l} = \{N_{m;p}(x_1)N_{n;p}(x_2)\}_{m=k-p,\dots,k;n=l-p,\dots,l} \tag{4.4}$$

so, it is necessary to compute their values at Gaussian quadrature integration points.

4.2. Selection of tasks and construction of tasks graph for the integration algorithm

4.2.1. Linear basis functions

Let us focus first on the linear basis functions case. We identify basic computational tasks for the integration algorithm and construct the alphabet of defined tasks as the computations performed by these tasks on actual data, in our case, on the integration points. The analysis presented in this section follows the theoretical plot given by the trace theory [35].

Definition 1. *The alphabet of tasks for linear B-spline based basis functions consists of the particular computational tasks executed during the integration process for linear B-spline basis functions, for a given data, namely for a given integration point:*

- *Computations of the entries of the frontal matrix, e.g.* $t^1_{k,l;k,l} = (B_{k,l;1}, B_{k,l;1})$ *as expressed in Table 4.2,*
- *Computations of the values of linear basis functions at Gaussian integration points, e.g.* $t^1_{k,l} = B_{k,l;1}(x_1, x_2)$ *as expressed in Table 4.3,*
- *Computations of the values of one dimensional B-spline basis functions values at Gaussian integration points, e.g.* $t^1_k = N_{k;1}(x_1)$ *as expressed in Table 4.4.*

The generation of the frontal matrix involves the computation of the values of scalar products $(B_{k,l;1}, B_{m,n;1})$ for $k, m = 1, \ldots, N_x; l, n = 1, \ldots, N_y$. It involves evaluation of the values of the multiplication of the two basis functions at Gaussian integration points. This operation is denoted by $t^1_{k,l;k,l} = B_{k,l;1}(x_1, x_2)B_{m,n;1}(x_1, x_2)$. This operation, in turn, can be expressed as multiplication of two operations, namely computing the value of $B_{k,l;1}(x_1, x_2; 1)$ and $B_{m,n;1}(x_1, x_2)$. We have denoted these basic operations as $t^1_{k,l}$ and $t^1_{m,n}$. Finally, evaluation of the value of two-dimensional B-spline at Gaussian quadrature point can be expressed as multiplication of $N_{k;1}(x_1)$ and $N_{l;1}(x_2)$. We have denoted these tasks by t^1_k, t^1_l. We can plot the graph presenting the dependency between these tasks. The graph presented in Figure 4.4 can be understood as a Dickert graph in the sense of the trace theory (compare [35]). The graph can be obtained by considering the representation of a trace, in the sense of the action of

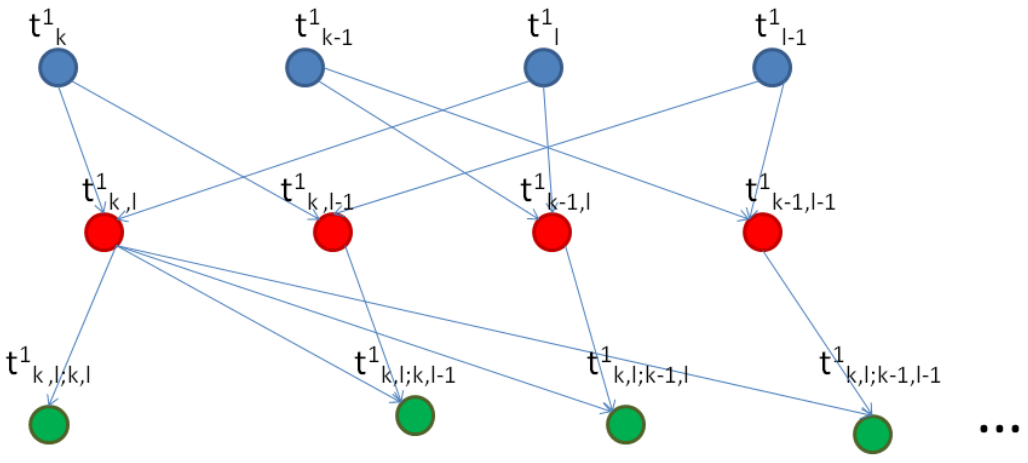


Figure 4.4: Dickert graph between tasks expressing the integration with linear basis functions

computation of the value at prescribed Gaussian quadrature point. By symbol "..." we denote the fact that we present only a small part of the graph. This is because the graph is huge since it involves all the Gaussian quadrature points and all the basis functions.

Finally, by executing the coloring of the Dickert graph (compare [35]), we obtain the sets of tasks that can be performed in parallel. In particular, all the 16 tasks can be executed concurrently.

4.2.2. Higher order basis functions

For higher order basis functions, the analysis is similar to the one performed for the linear case. However, we now need the help of the recursive Cox-de-Boor formulae, presented in Figure 4.5, expressing the higher order B-splines as a linear combination of lower order B-splines. By using this formulae, we can express the higher order B-splines as multiplications and additions of lower order B-splines and extend our analysis to higher order cases.

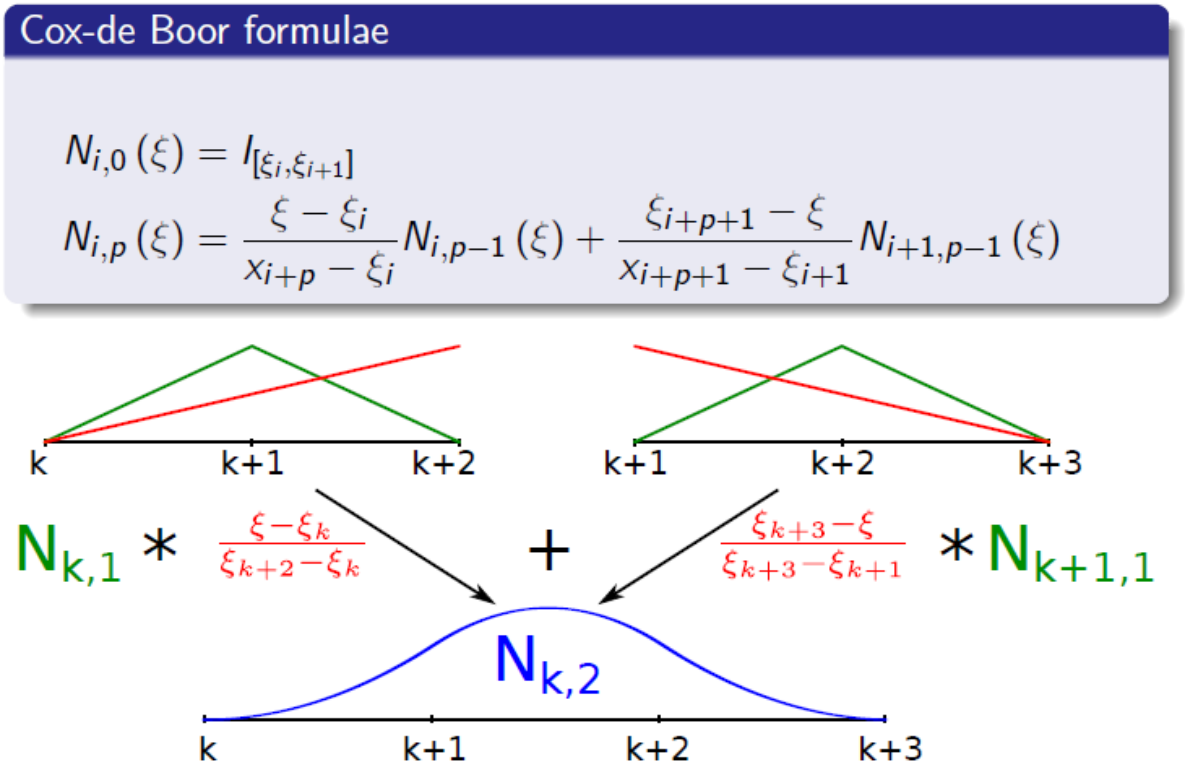


Figure 4.5: Cox-de-Boor formulae

Let us focus on the quadratic basis functions case and perform the task identification again, utilizing the Cox-de-Boor formulae. In other words, we identify the alphabet of tasks, understood as computational tasks performed at given integration points.

Definition 2. *The alphabet of tasks for quadratic B-spline based basis functions consists of:*

- *Computations of the entries of the frontal matrix, e.g.* $t^2_{k,l;k,l} = (B_{k,l;2}, B_{k,l;2})$ *as expressed in Table 4.6,*
- *Computations of the values of quadratic basis functions at Gaussian integration points, e.g.* $t^2_{k,l} = B_{k,l;2}(x_1, x_2)$ *as expressed in Table 4.7,*
- *Computations of the values of one dimensional second order B-spline basis functions at Gaussian integration points, e.g.* $t^2_k = N_{k;2}(x_1)$ *as expressed in Table 4.8.*

– *Computations of the values of one dimensional first order B-spline basis functions at Gaussian integration points, e.g.* $t_k^1 = N_{k;1}(x_1)$ *as expressed in Table 4.4.*

Again, we introduce the dependency relation between tasks and display a Dickert graph (compare [35]), see Figure 4.6. At the last level, there are $9 * 9 = 81$ tasks, and we only show 9 of them for simplicity of the presentation. The Dickert graph can be colored to obtain the sets of tasks that can be executed in parallel, including all the 81 tasks from the last level in one final set. Now, the parallel integration can be performed with the scheduling according to colors of the tasks.

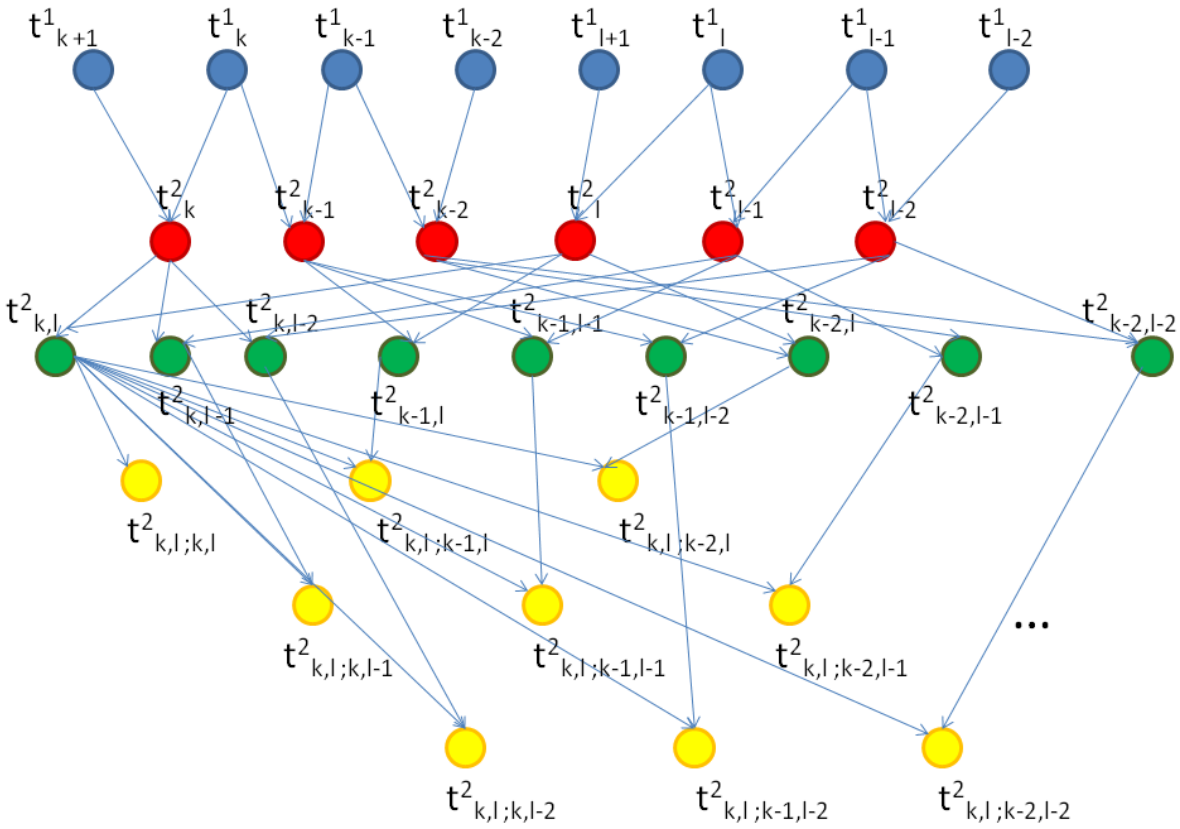


Figure 4.6: Dickert graph between tasks expressing the integration with quadratic basis functions

4.3. Parallel OpenMP implementation

The standard algorithm for integration and aggregation in all the mentioned cases is identical. In general, the generation of the matrices for finite element method computations involves nested loops, starting from the elements, Gaussian integration points, through test basis functions, to trial basis functions. Our parallelization of the integration process is based on the decomposition of loops concerning local basis functions and Gaussian quadrature points. Below we present the OpenMP pseudo-code algorithm responsible for the integration of the element matrices.

```
use omp_lib

!$OMP PARALLEL DO
```

```
!$OMP& DEFAULT (SHARED)
!$OMP& FIRSTPRIVATE(ix,ex,ey,ez,tempA,nr_nonzeros,J,ax,ay,az,
!$OMP& ax1,ay1,az1,ind,ind1,W,value,kx,ky,kz,threadID)
      do iy=1,miy
        ez=modulo(iy-1,nelemz)
        ix=(iy-ez)/nelemz+1
        ey=modulo(ix-1,nelemy)
        ex=(ix-ey)/nelemy+1
        threadID=OMP_GET_THREAD_NUM()+1
        if(associated(tempA)) then; else
          tempA=>AllArr(threadID)
          nr_nonzeros=0
        endif
        J = Jx(ex)*Jy(ey)*Jz(ez)
        do ax = 0,px
          do ay = 0,py
            do az = 0,pz
              do ax1 = 0,px
                do ay1 = 0,py
                  do az1 = 0,pz
                    ind = (Ox(ex)+ax)+(Oy(ey)+ay)*(nx+1)+
     .                (Oz(ez)+az)*(ny+1)*(nx+1)+1
                    ind1 =(Ox(ex)+ax1)+(Oy(ey)+ay1)*(nx+1)+
     .                (Oz(ez)+az1)*(ny+1)*(nx+1)+1
                    if(ind.gt.ind1)cycle
                    nr_nonzeros=nr_nonzeros+1
                    tempA%IRN(nr_nonzeros)=ind
                    tempA%JCN(nr_nonzeros)=ind1
                    value = 0.d0
                    do kx = 1,ngx
                      do ky = 1,ngy
                        do kz = 1,ngz
                          W = Wx(kx)*Wy(ky)*Wz(kz)*J
                          value = value +
     .                      NNx(0,ax,kx,ex)*
     .                        NNy(0,ay,ky,ey)*
     .                        NNz(0,az,kz,ez)*W *
     .                        NNx(0,ax1,kx,ex)*
     .                        NNy(0,ay1,ky,ey)*
     .                        NNz(0,az1,kz,ez)
                        end do
                      end do
                    end do
                    tempA%A(nr_nonzeros)=value
                    if(tempA%arrsize.eq.nr_nonzeros) then
```

```
                        tempA%i=nr_nonzeros
                        allocate(tempA%next)
                        tempA=>tempA%next
                        allocate(tempA%A(nr_nonzeros))
                        allocate(tempA%IRN(nr_nonzeros))
                        allocate(tempA%JCN(nr_nonzeros))
                        tempA%A = 0.d0
                        tempA%arrsize=nr_nonzeros
                        tempA%i = 0
                        nr_nonzeros=0
                      endif
                    end do
                  end do
                end do
              end do
            end do
          end do
          tempA%i=nr_nonzeros
        end do
!$OMP END PARALLEL DO
```

Chapter 5

Numerical results

The main goal of this chapter is to present numerical scalability of algorithms presented in this book. We measure computational cost of algorithms presented in chapters: 2, 3 and 4. We refer to section 1.3.5 for results of simulation from section 5.6.

5.1. Classical parallel multi-frontal solver for distributed memory cluster

The goal of this section is to verify the computational complexities for parallel version of multi-frontal solver derived in chapter 2 in equation 2.10. The numerical experiments were performed on STAMPEDE [76], a Linux cluster hosted by the Texas Advanced Computing Center. The simulations were performed with PETIGA [18], and utilized parallel MUMPS solver [5, 6, 7] with parallel Scalapack [21] dense solver, parallel SuperLU solver [58, 59], and parallel PaStiX solver [51] available from PETSc library [9, 10, 11]. In the experiments, we utilized one core per Linux cluster node to maximize the amount of available memory per node. The only exception are the two experiments with 512 cores for linear B-splines in three-dimensional case, where we utilize two cores per node.

5.1.1. Two-dimensional case

The experiments have been performed, for mesh sizes with 128^2, 256^2, 512^2, 1024^2 and 2048^2 elements, for some numbers of processors from 1 to 256. It is not possible to solve any larger problem for two-dimensional IGA on STAMPEDE with direct solvers since all the solvers run out of memory. Alternative option would be to go for out-of-core, but this slows down the solution time by order of magnitude [66].

Weak scaling efficiency

We start illustrating the weak scaling efficiency for the three different direct solvers executed for two-dimensional 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 Figures 5.1-5.3, for MUMPS, PasTiX and SuperLU, for linear, quartic and octic B-splines, with C^0, C^3 and C^7 global continuity, respectively.

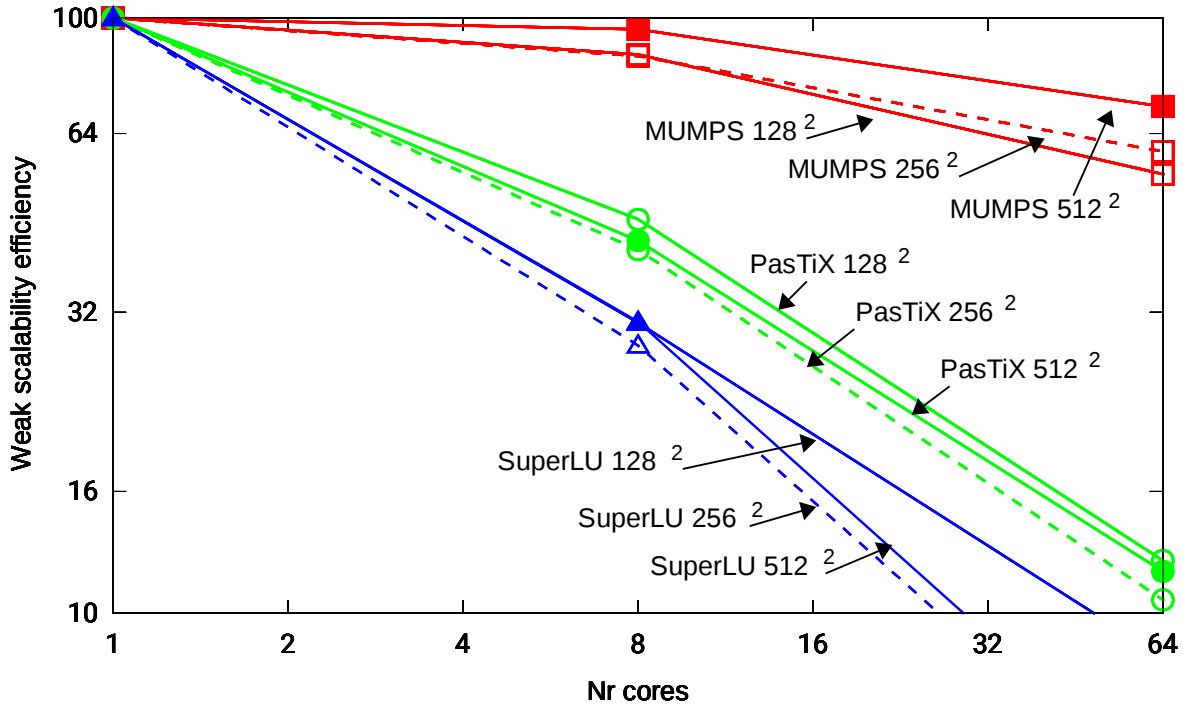


Figure 5.1: Weak scaling efficiency of the direct solvers for two-dimensional IGA with linear B-splines, C^0 global continuity.

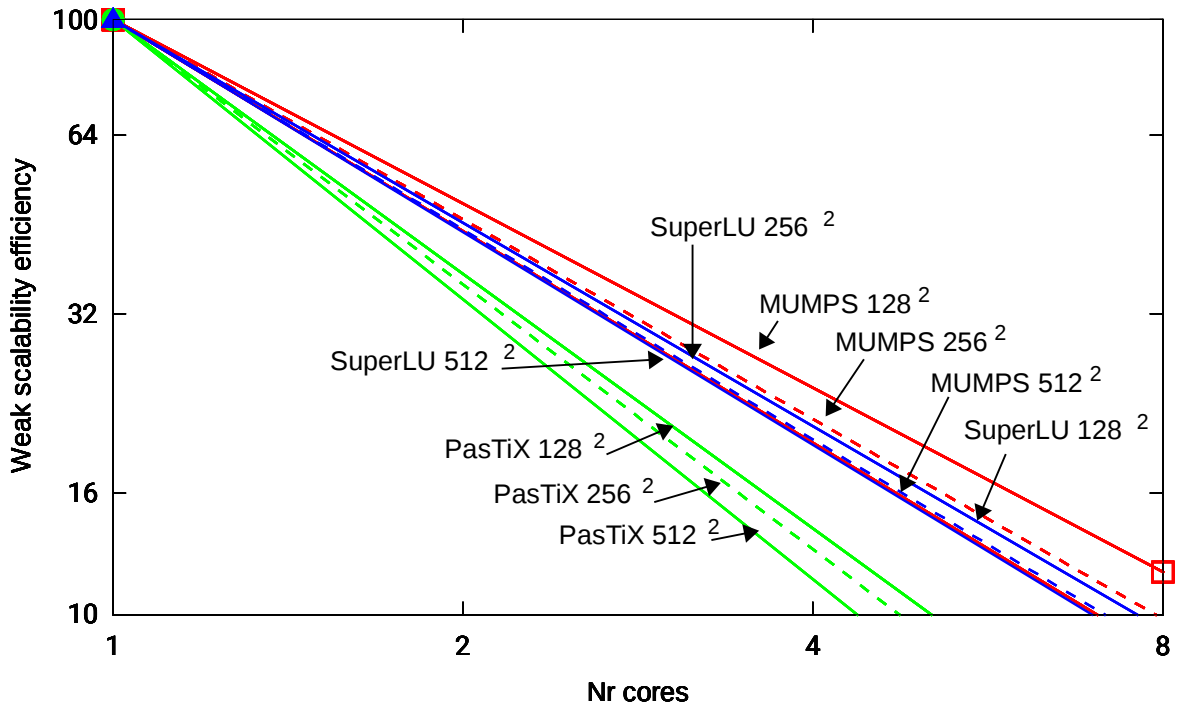


Figure 5.2: Weak scaling efficiency of the direct solvers for two-dimensional IGA with quartic B-splines, C^3 global continuity.

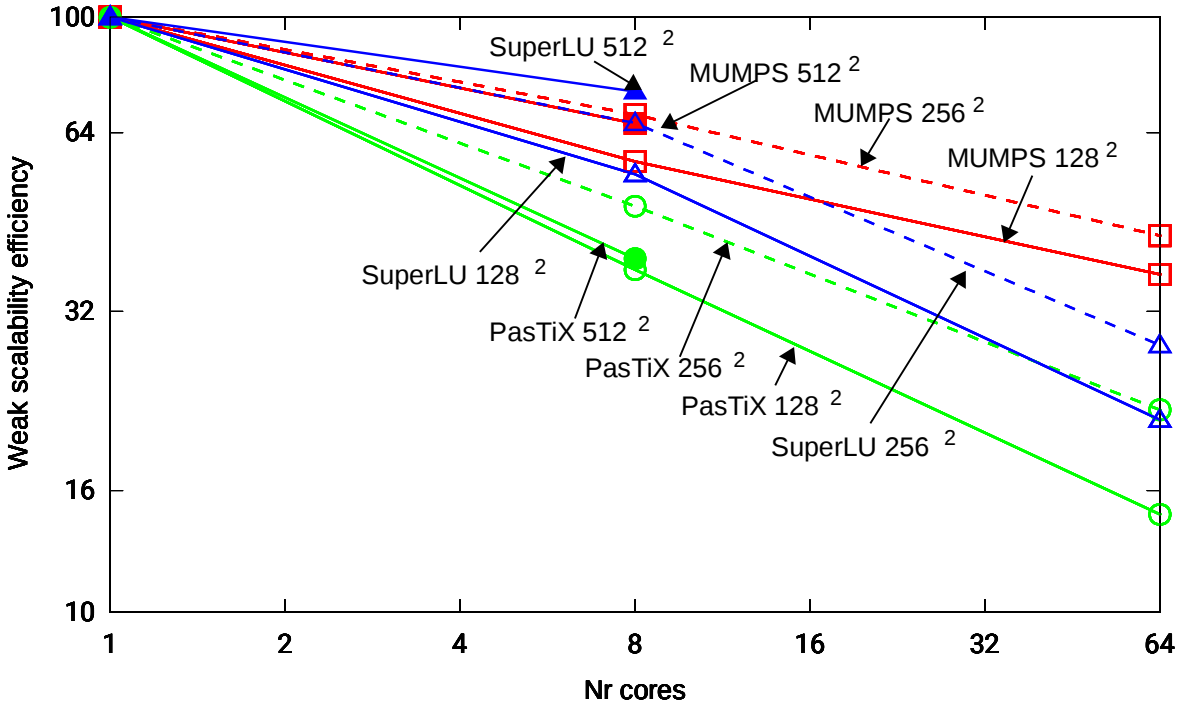


Figure 5.3: Weak scaling efficiency of the direct solvers for two-dimensional IGA with octic B-splines, C^7 global continuity.

5.1.2. Strong scaling efficiency

Let us focus now on strong scaling efficiency of the MUMPS solver executed for two-dimensional 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 Figures 5.4, 5.5 and 5.6, 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.

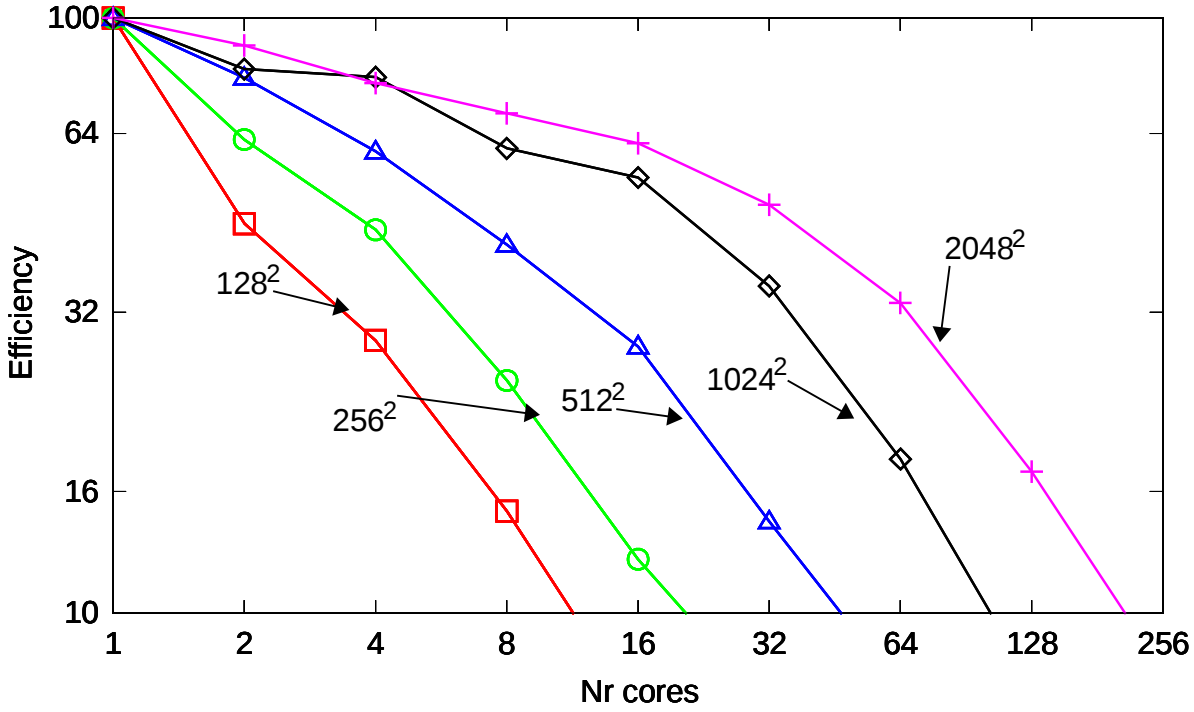


Figure 5.4: Parallel efficiency of the MUMPS direct solver for two-dimensional IGA with linear B-splines, with C^0global continuity.

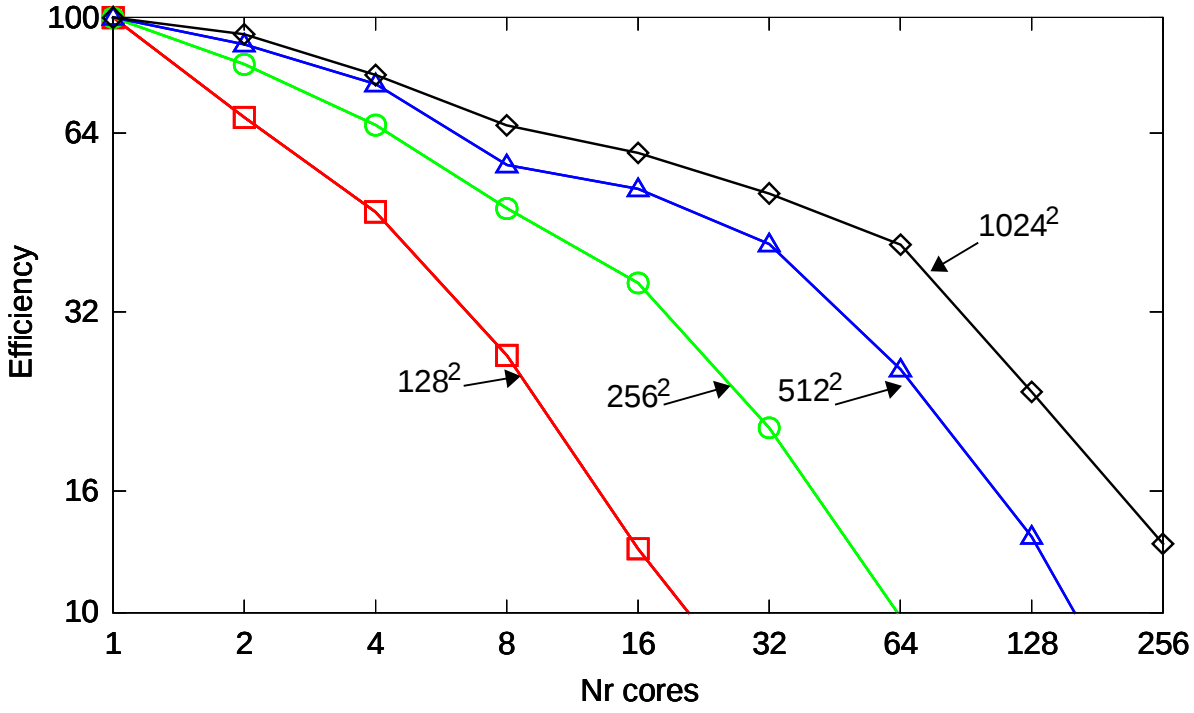


Figure 5.5: Parallel efficiency of the MUMPS direct solver for two-dimensional IGA with quartic B-splines, with C^3 global continuity.

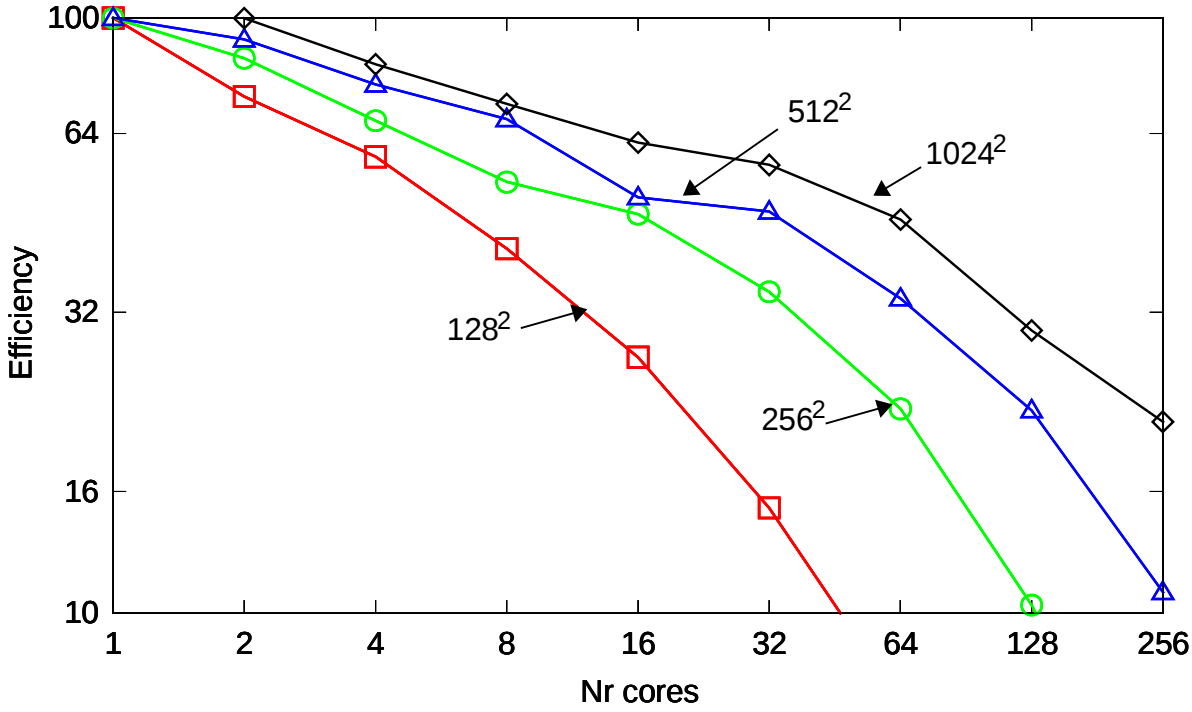


Figure 5.6: Parallel efficiency of the MUMPS direct solver for two-dimensional IGA with octic B-splines, with C^7 global continuity.

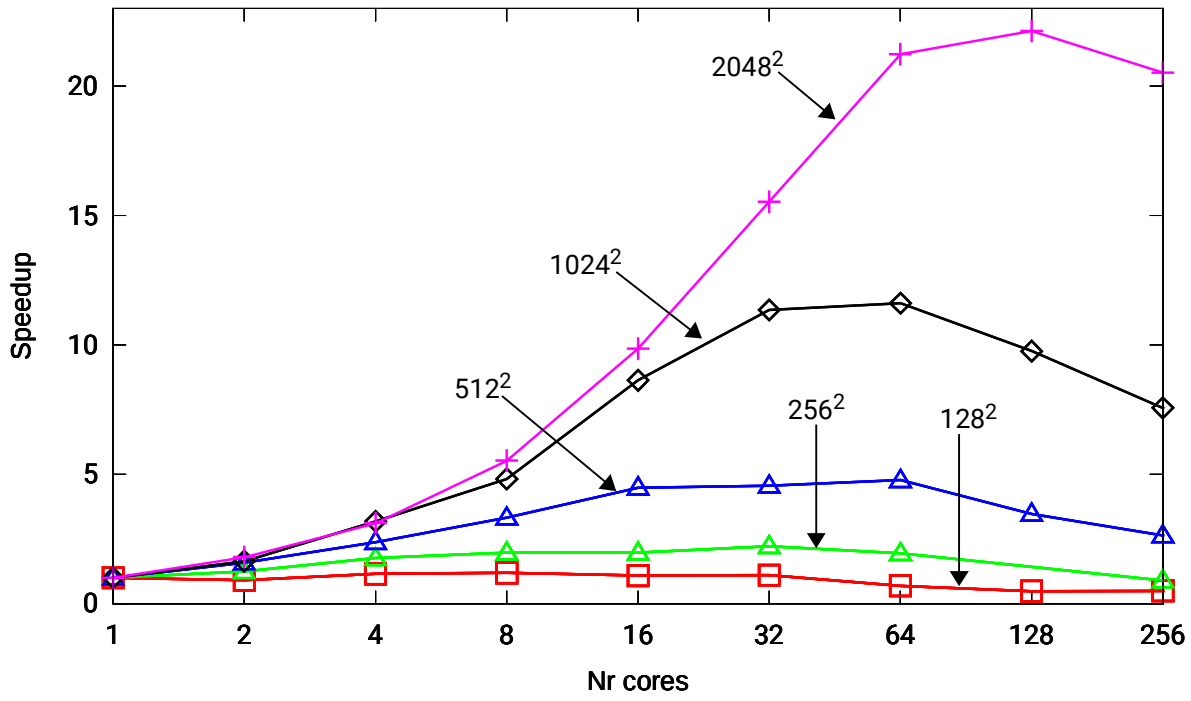


Figure 5.7: Parallel speedup of the MUMPS direct solver for two-dimensional IGA with linear B-splines, with C^0global continuity.

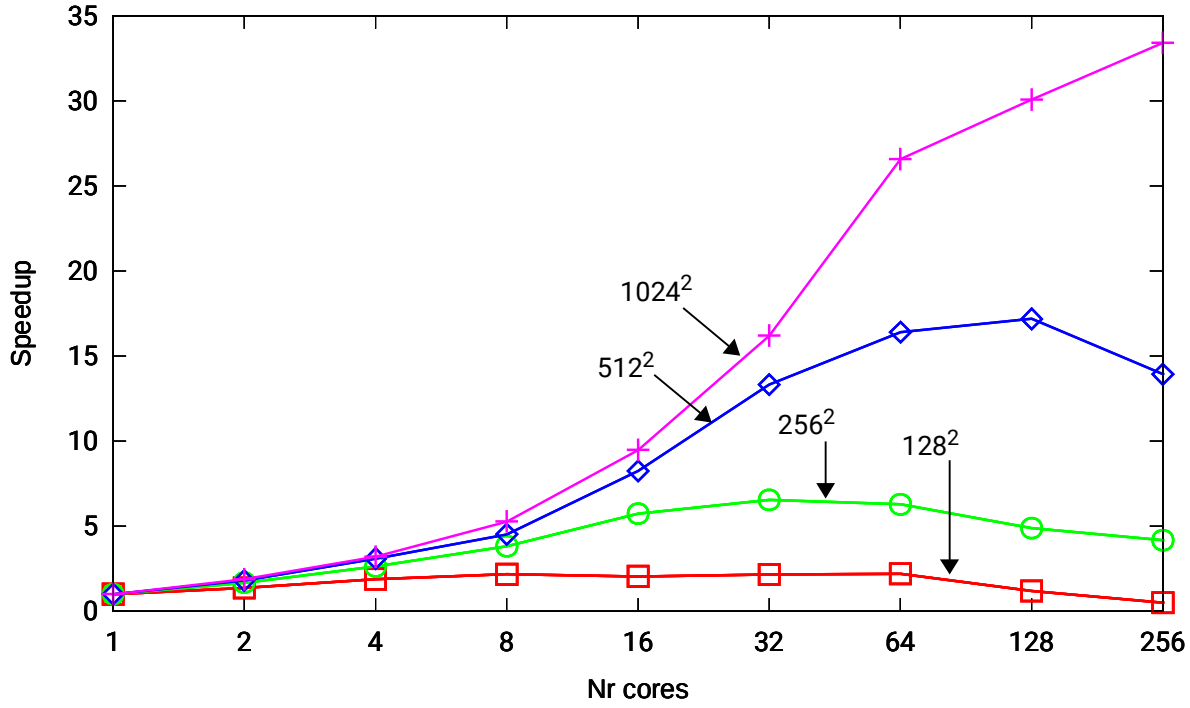


Figure 5.8: Parallel speedup of the MUMPS direct solver for two-dimensional IGA with quartic B-splines, with C^3 global continuity.

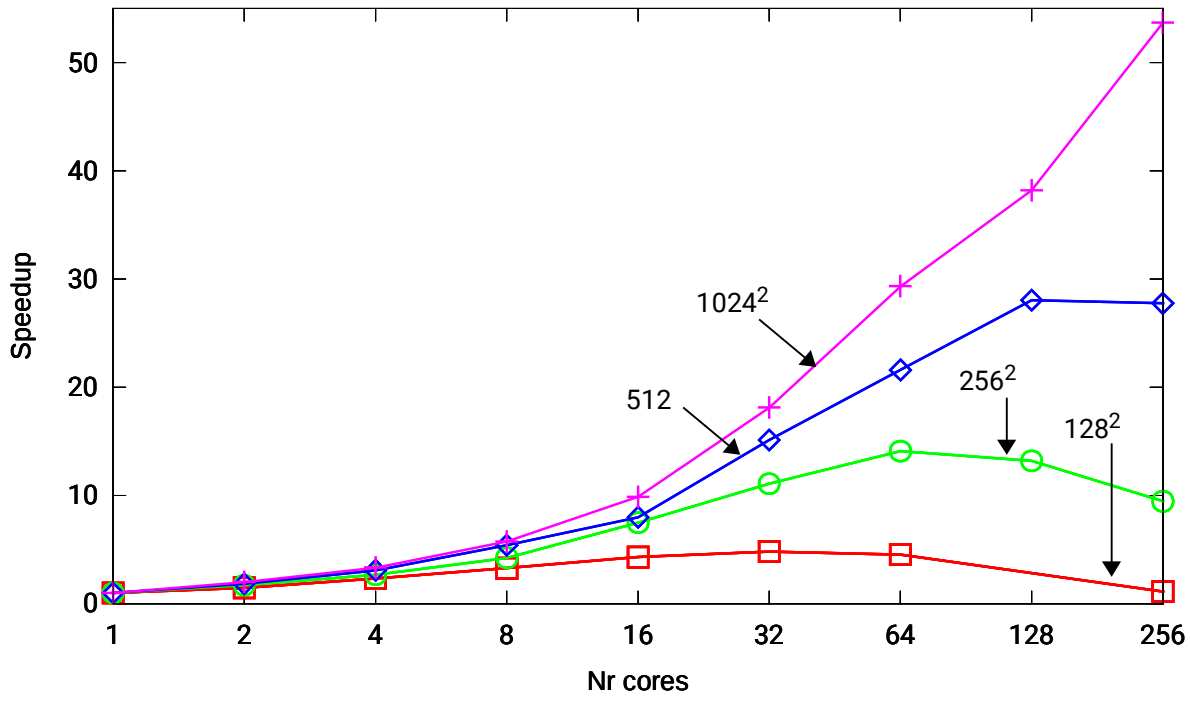


Figure 5.9: Parallel speedup of the MUMPS direct solver for two-dimensional IGA with octic B-splines, with C^7 global continuity.

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

Figures 5.10 and 5.11 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 the 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, solvers an ascending line denotes a growth worse than p^2. As predicted by our model, the solver's 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 two-dimensional IGA model problem, for increasing problem size N and 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 Figure 5.12 for linear B-splines and C^0 global continuity; in Figure 5.13 for quartic B-splines and C^3 global continuity; in Figure 5.14 for octic B-splines and C^7 global continuity.

The curve fitting algorithm estimated the α exponent factor as summarized in Table 5.1 for linear B-splines, in Table 5.2 for quartic B-splines and in Table 5.3 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.

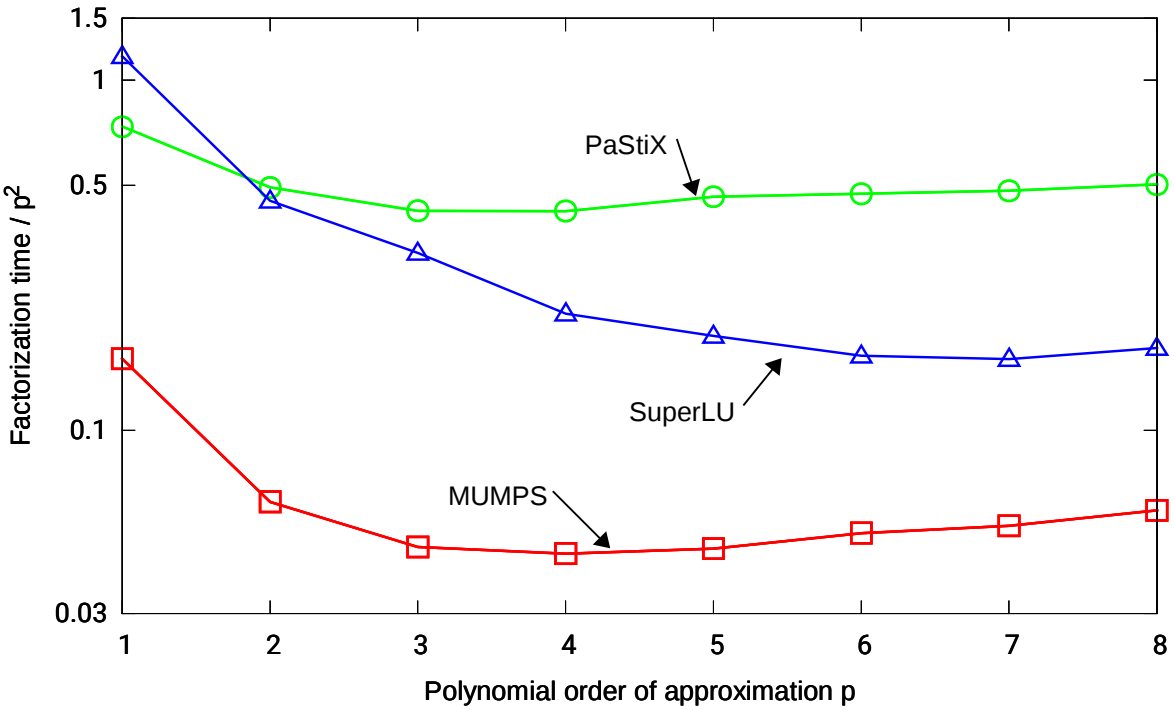


Figure 5.10: Execution time divided by p^2 measured for parallel MUMPS, SuperLU and PaStiX solvers executed over distributed memory machine, for two-dimensional problem with 256×256 elements, for continuity C^{p-1} for different p, one core per node, with eight nodes.

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

Table 5.1: Exponent factors α from fitting the curve $const * N^{\alpha}$ based on execution times of MUMPS solver for two-dimensional IGA with linear B-splines, C^0 continuity.

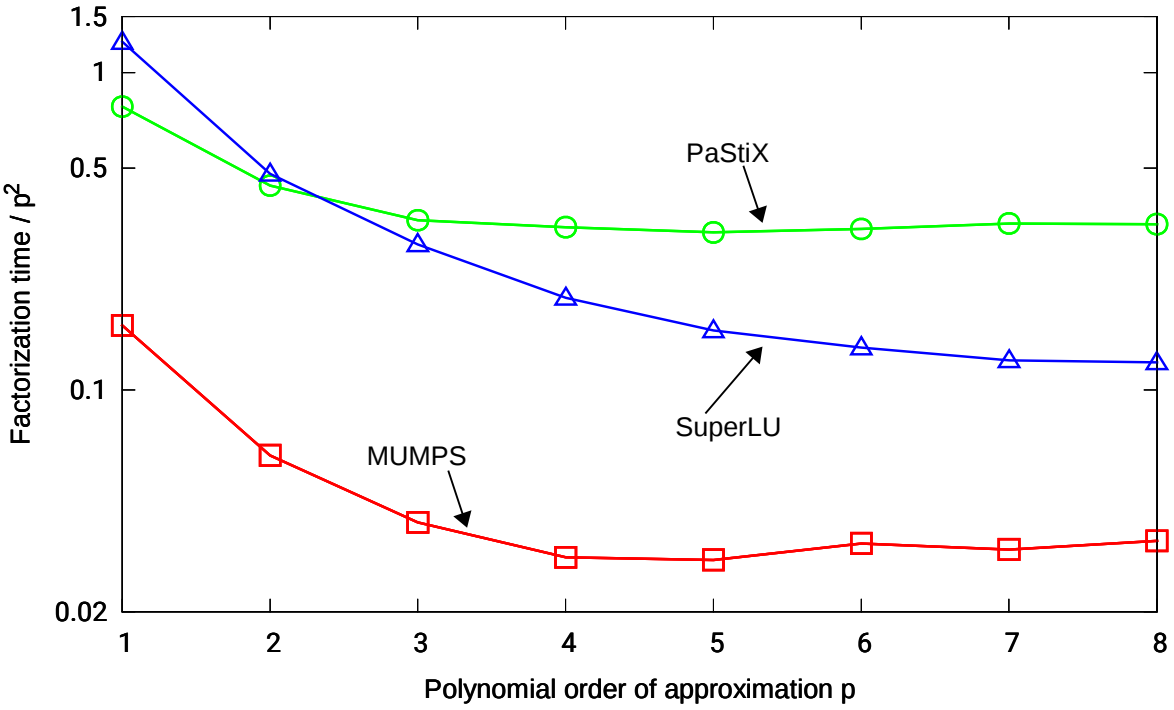


Figure 5.11: Execution time divided by p^2 measured for parallel MUMPS, SuperLU and PaStiX solvers executed over distributed memory machine, for two-dimensional problem with 256×256 elements, for continuity C^{p-1} for different p, one core per node, with thirty two nodes.

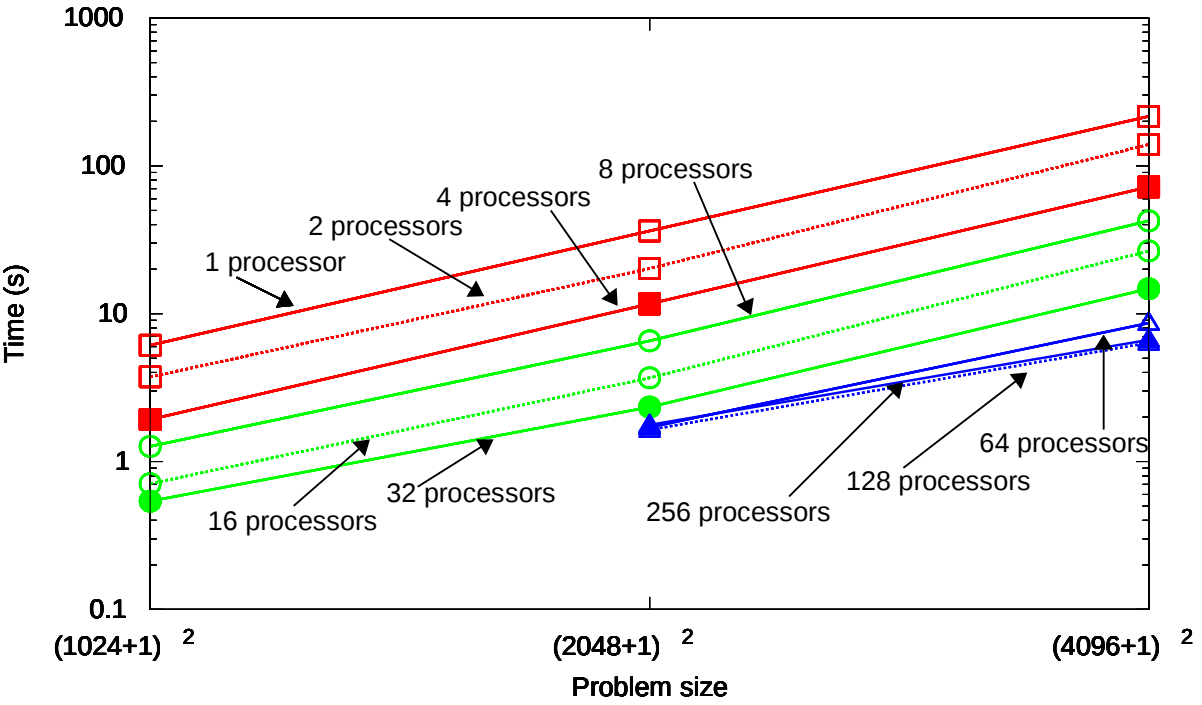


Figure 5.12: Execution times for the MUMPS direct solver for two-dimensional IGA with linear B-splines, C^0 global continuity.

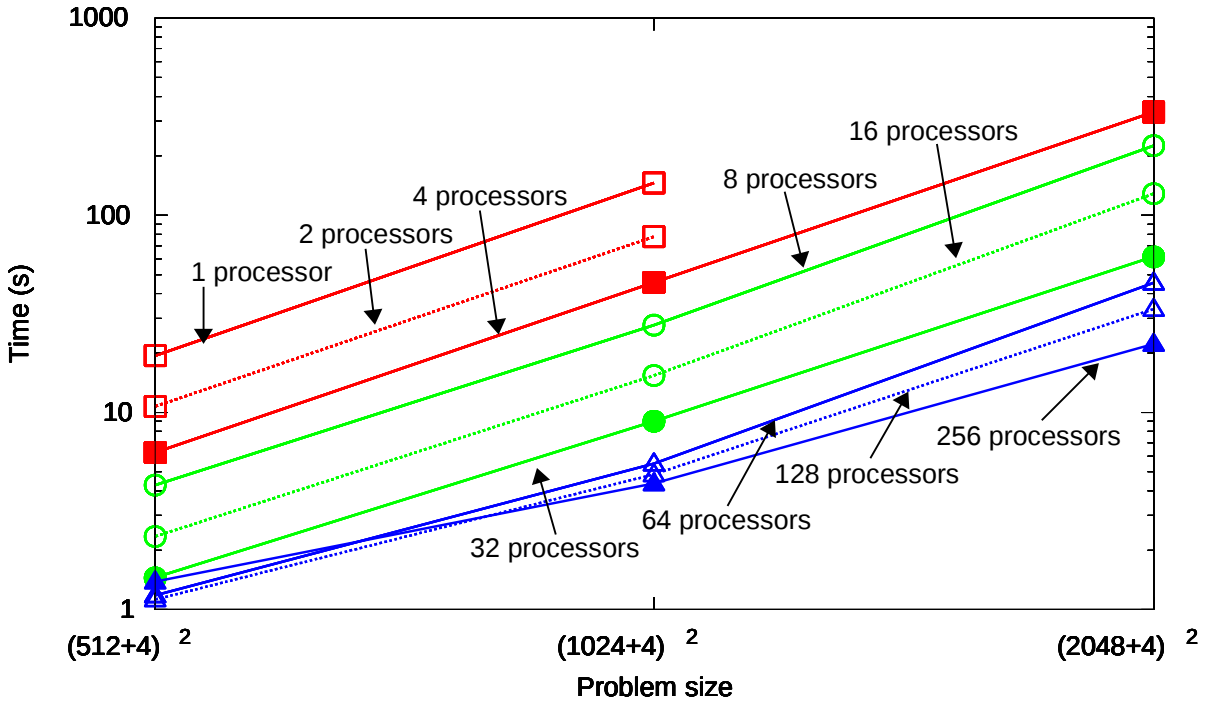


Figure 5.13: Execution times for the MUMPS direct solver for two-dimensional IGA with quartic B-splines, C^3 global continuity.

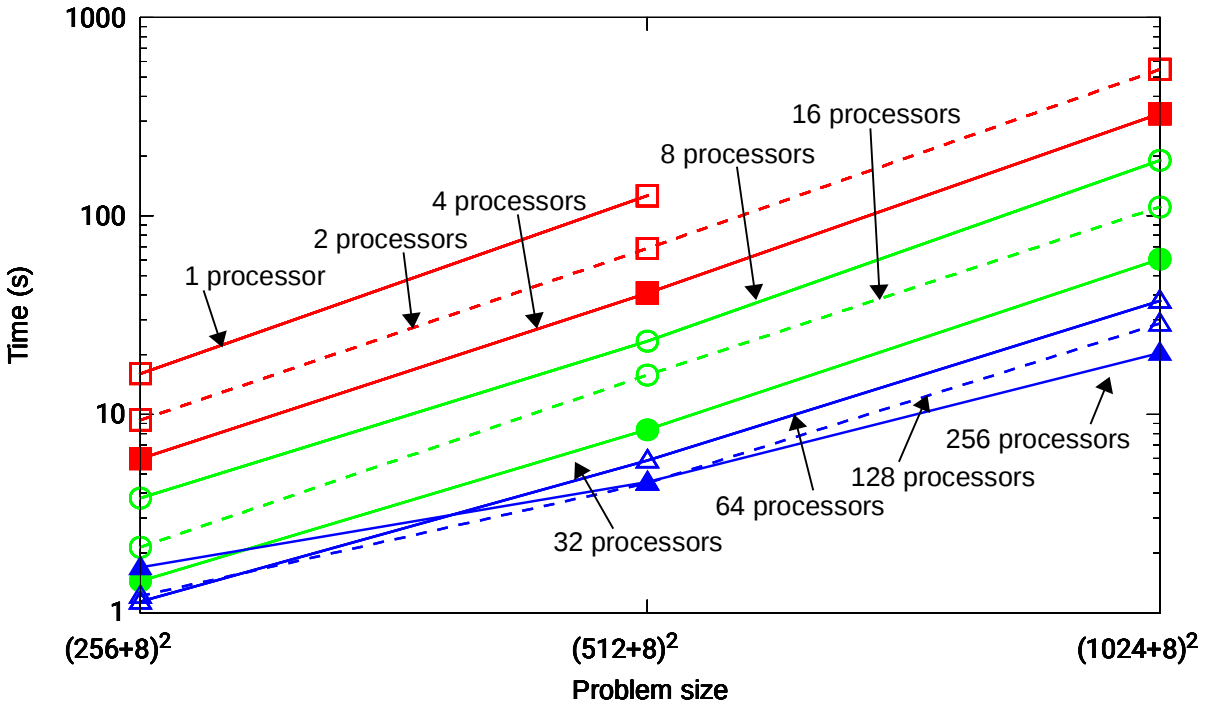


Figure 5.14: Execution times for the MUMPS direct solver for two-dimensional IGA with octic B-splines, C^7 global continuity.

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 5.2: Exponent factors α from fitting the curve $const * N^{\alpha}$ based on execution times of MUMPS solver for two-dimensional IGA with quartic B-splines, C^3 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

Table 5.3: Exponent factors α from fitting the curve $const * N^{\alpha}$ based on execution times of MUMPS solver for two-dimensional IGA with octic B-splines, C^7 continuity.

5.1.4. Three-dimensional case

Weak scaling efficiency

We start illustrating the weak scaling efficiency of MUMPS solver executed for three-dimensional IGA problem. The weak scaling efficiency results are presented in Figures 5.15 and 5.16 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.

For octic B-splines with C^7 continuity, we run out of memory.

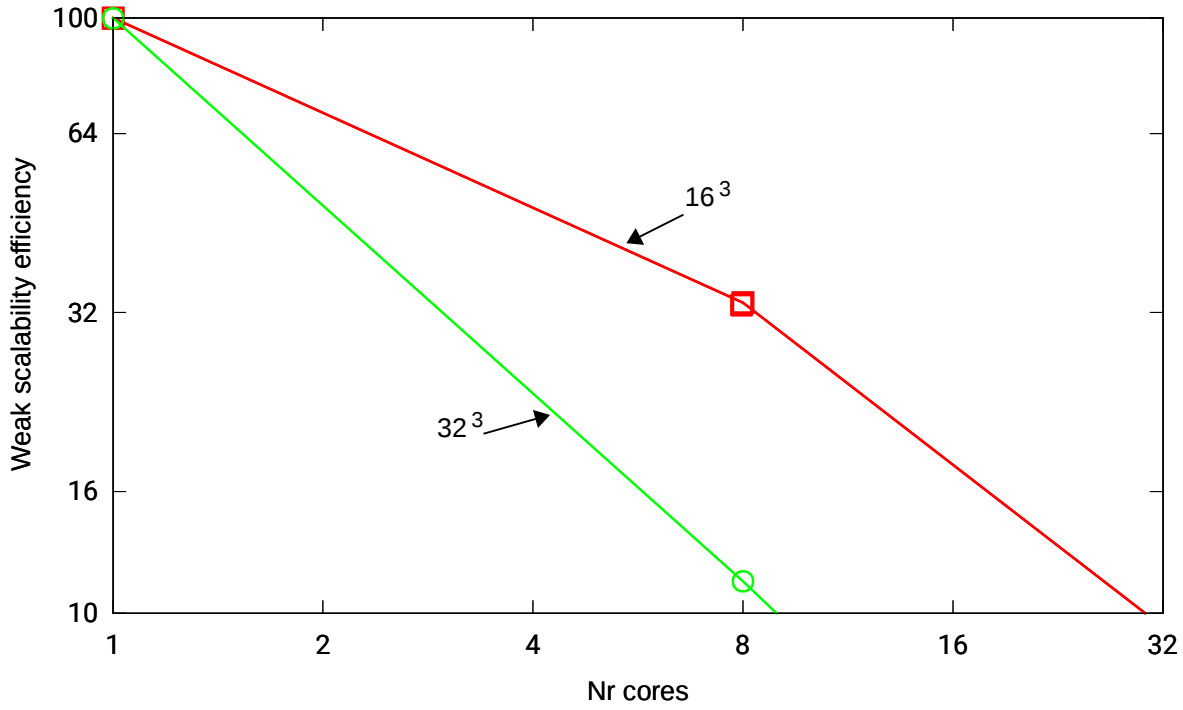


Figure 5.15: Weak scalability of the MUMPS direct solver for three-dimensional IGA with linear B-splines, C^0 global continuity. Different lines correspond to various problems sizes.

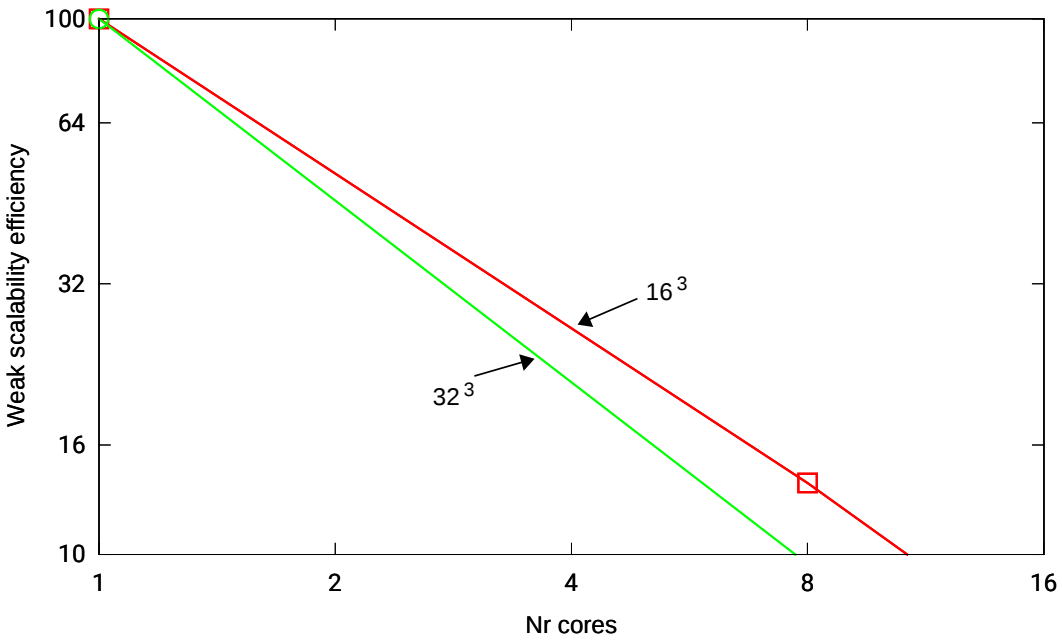


Figure 5.16: Weak scalability of the MUMPS direct solver for three-dimensional IGA with quartic B-splines, C^3 global continuity. Different lines correspond to various problems sizes.

Strong scaling efficiency

Let us focus now on strong scaling efficiency of the MUMPS solver executed for three-dimensional IGA problem. The efficiency results are presented in Figures 5.17 and 5.18, for linear and quartic B-splines, with C^0 and C^3 global continuity, respectively. The speedup results are presented in Figures 5.19 and 5.20, 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.

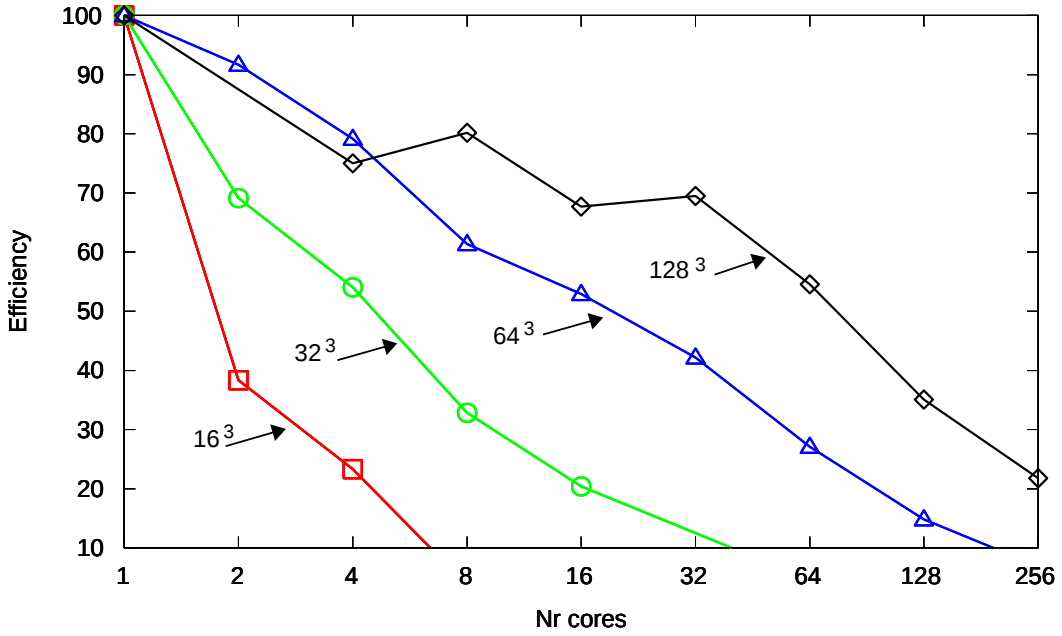


Figure 5.17: Parallel efficiency of the MUMPS direct solver for three-dimensional IGA with linear B-splines, C^0 global continuity. Different lines represents different sizes of the mesh.

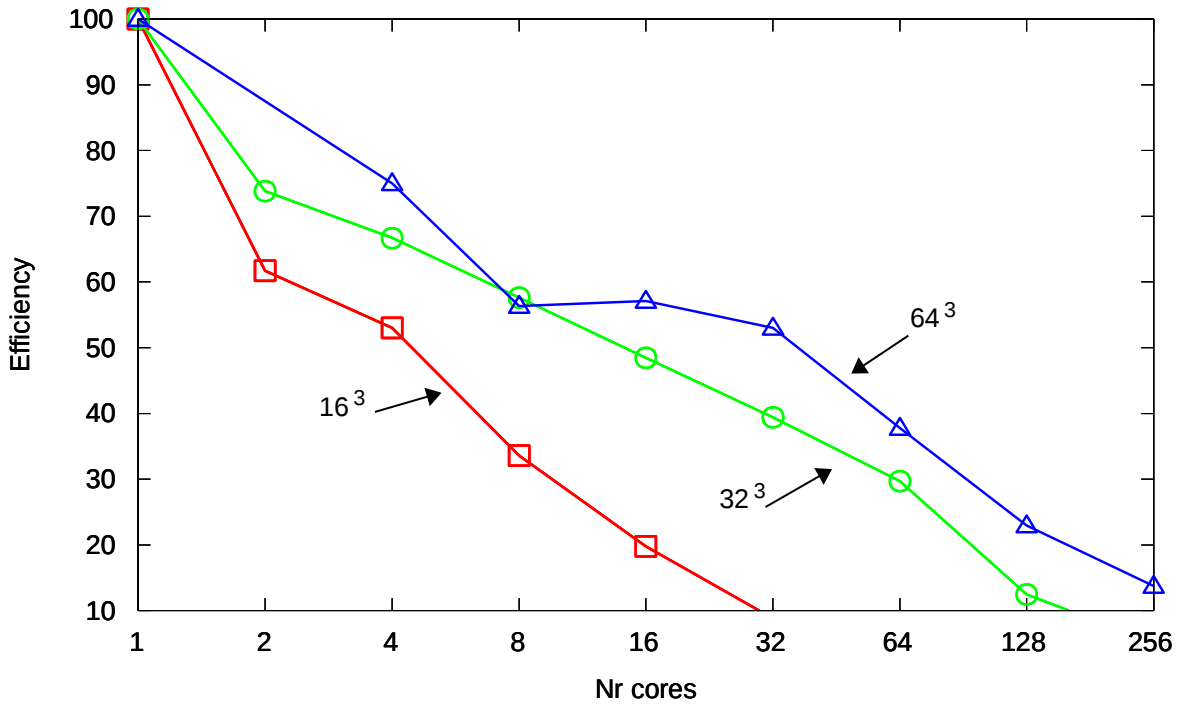


Figure 5.18: Parallel efficiency of the MUMPS direct solver for three-dimensional IGA with quartic B-splines, C^3 global continuity. Different plots represents different sizes of the mesh.

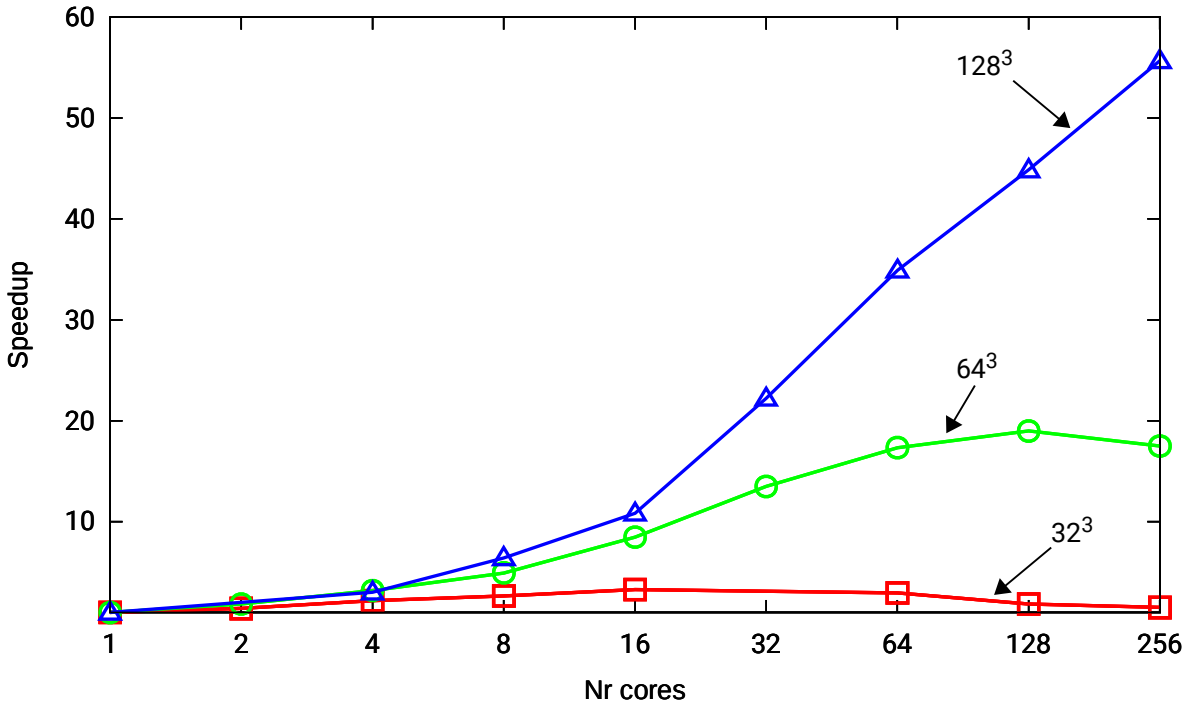


Figure 5.19: Parallel speedup of the MUMPS direct solver for three-dimensional IGA with linear B-splines, C^0 global continuity. Different lines represents different sizes of the mesh.

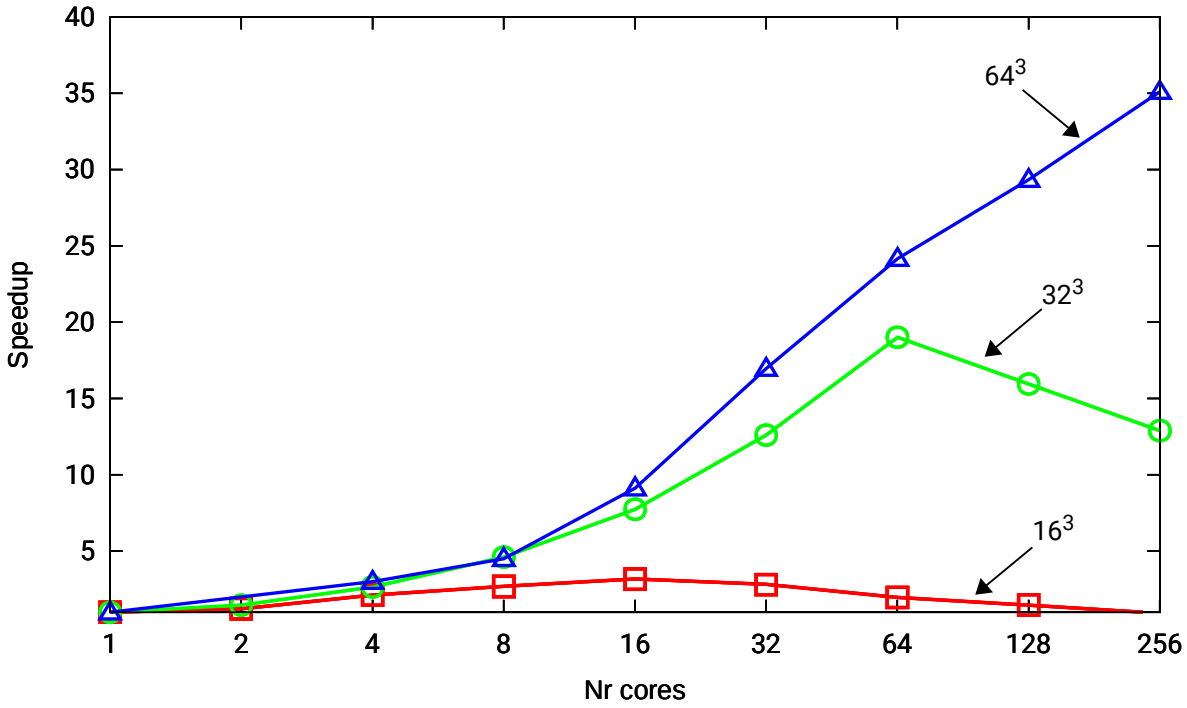


Figure 5.20: Parallel speedup of the MUMPS direct solver for three-dimensional IGA with quartic B-splines, C^3 global continuity. Different plots represents different sizes of the mesh.

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

We display the execution times of the parallel MUMPS solver for the three-dimensional IGA model problem, as we increase problem size N for fixed p, moreover, 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 Figure 5.21 for linear B-splines with C^0 global continuity; in Figure 5.22 for quartic B-splines with C^3 global continuity.

The curve fitting algorithm estimated the α exponent factor as summarized in Table 5.4 for linear B-splines, and in Table 5.5 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.

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

Finally, we display the execution times of the parallel MUMPS solver for the three-dimensional IGA model problem, for increasing continuity p and fixed N, divide the execution time by $N^{4/3}$, moreover, execute the curve fitting algorithm to estimate the exponent factor in formula $\mathcal{O}\left(const * p^{\beta}\right)$. The execution times as a function of p are presented in Figure 5.23 for quartic B-splines, C^3 global continuity.

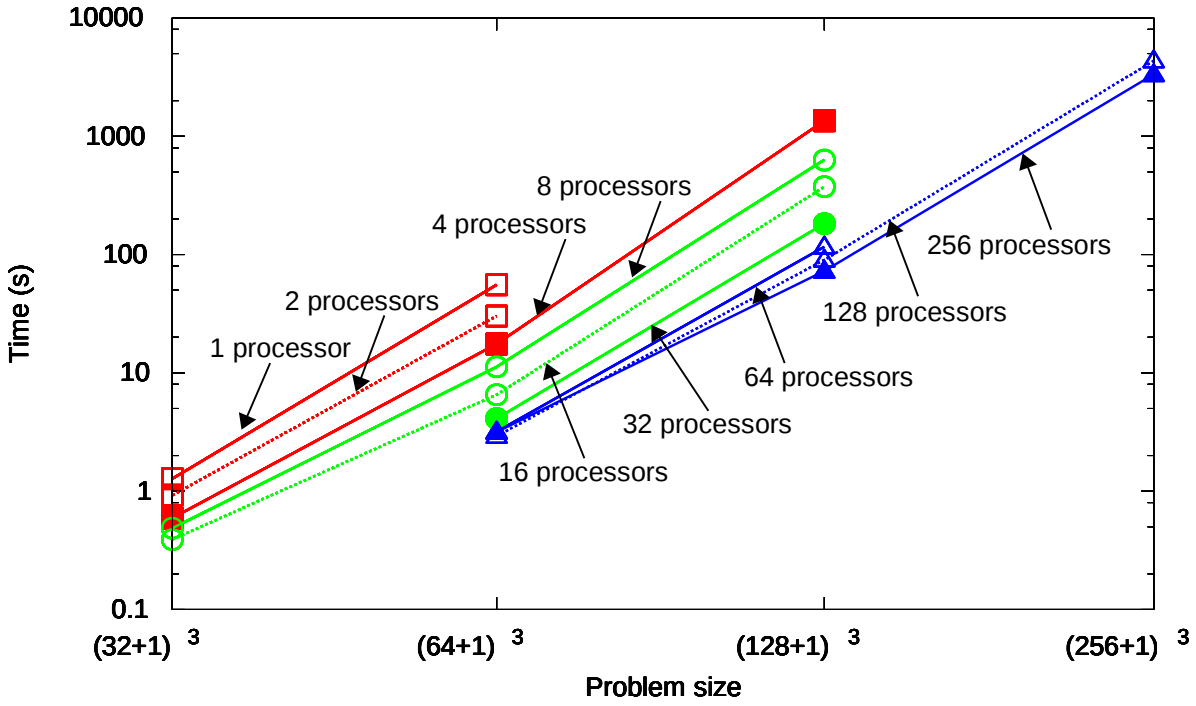


Figure 5.21: Execution times for the MUMPS direct solver for three-dimensional IGA with linear B-splines, C^0 global continuity.

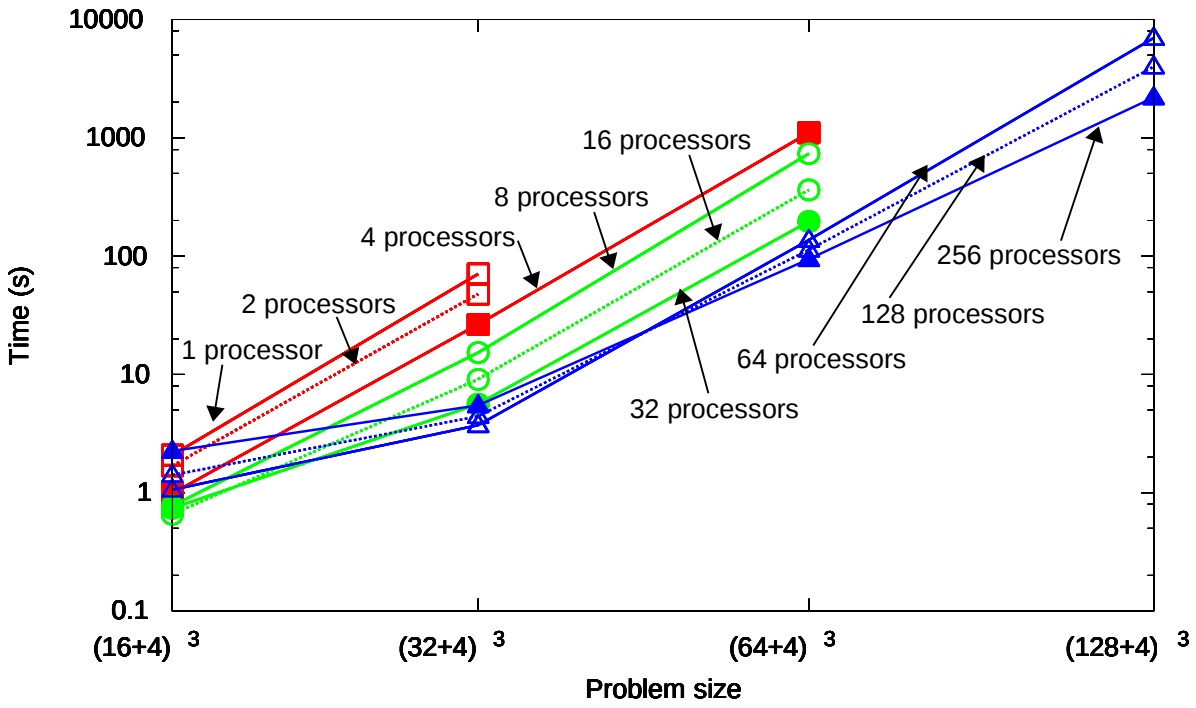


Figure 5.22: Execution times for the MUMPS direct solver for three-dimensional IGA with quartic B-splines, C^3 global continuity.

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

Table 5.5: Exponent factors α from fitting the curve $const * N^{\alpha}$ based on execution times of MUMPS solver for three-dimensional IGA with quartic B-splines, C^3 continuity.

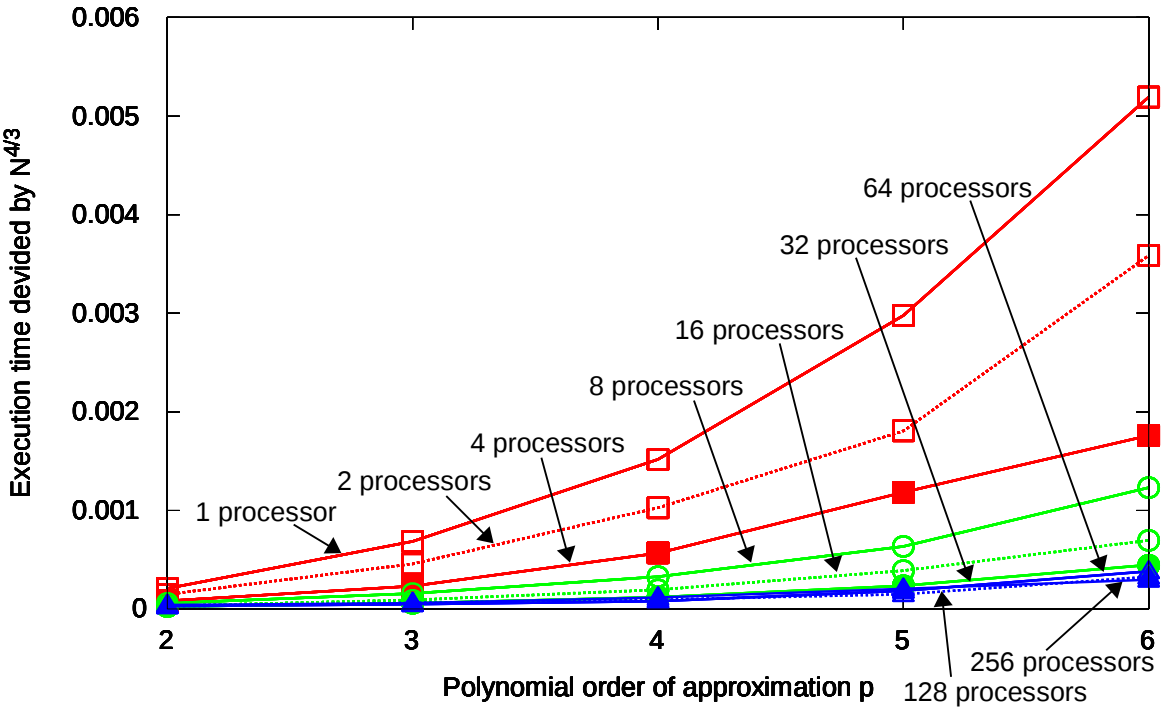


Figure 5.23: Execution times for the MUMPS direct solver for three-dimensional IGA, for fixed 32^3 elements, divided by $N^{4/3}$.

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

Table 5.6: Exponent factors β from $const * p^{\beta}$ curve fitting based on execution times of MUMPS solver for three-dimensional IGA with quartic B-splines, C^3 continuity, for fixed N, divided by$N^{4/3}$.

The curve fitting algorithm estimated the β exponent factor as summarized in Table 5.6 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.

5.2. Parallel isogeometric L^2 projection for distributed memory machines

The model has been verified by comparing with numerical experiments performed on the LONESTAR Linux cluster, with $N = 512$ or $N = 1024$ degrees of freedom in each direction, with $p = 3$. The comparison of the total execution time is given in Figures 5.24 and 5.25. We can draw the following conclusions from these Figures. There is a quite good agreement between the theoretical estimates and numerical experiments for both 512^3 and 1024^3 cases with cubic polynomials. Good scalability of the solver is maintained up to $1,000$ of processors. We can solve $134,217,728$ unknowns resulting from the three-dimensional cube of 512^3 elements with cubic B-splines within 20 seconds using 1000 processors. We can also solve $1,073,741,824$ unknowns resulting from the three-dimensional cube of 1024^3 elements with cubic B-splines within 3 minutes by using 1000 processors.

Figures 5.26 and 5.27 present the comparison of the experimental and theoretical integration times. We can draw the following conclusions from these Figures. Again, there is a quite good agreement between the theoretical estimates and experimental results for the integration execution time. The integration time is dominating the solution time significantly. In other words, the generation of the projection data takes much more time than the isogeometric L^2 projections using alternating direction solver itself, and it means that our solver algorithm performs very well (usually the solution takes much more time than integration). There is a need for our fast parallel trace theory based integration algorithm.

Figures 5.28 and 5.29 present the comparison of the experimental and theoretical solution times. We measure three solution phases, corresponding to steps 1b, 2b, and 3b of the general algorithm. We can draw the following conclusions from these Figures. There is still the quite good agreement between the theoretical estimates and experimental results for the solution times. The solution time takes around 1 percent of the total solver time. In our solver, we utilize multiple $1D$ sequential block diagonal multifrontal solvers with many right-hand sides working on the faces of the cube of three-dimensional processors. Possible improvement of the algorithm is to utilize parallel block-diagonal solvers (e.g. designed using the trace-theory based approach, working within rows of processors (10 processors per solver in 1000 processors case). However, the solution time contributes only 1% to overall computation time, and further improvement would not be beneficial.

Figures 5.30 and 5.31 present the comparison of the experimental and theoretical gather times. We measure three gathering phases, corresponding to steps 1a, 2a, and 3a of the general algorithm. We can draw the following conclusions from these Figures. There is a good agreement between the theoretical and experimental gather times for second and third gather. However, the first gather takes less time than predicted by the model. Our second and third gather times include the data reorder phase, while the first gather is just the communication itself.

Figures 5.32 and 5.33 present the comparison of the experimental and theoretical scatter times. We measure three scattering phases, corresponding to steps 1c, 2c, and 3c of the general algorithm. We can draw the following conclusions from these Figures. There is a quite good agreement between the theoretical and experimental scatter times for all the phases. The experimental scatters becoming little slower when we increase the number of processors. However, the difference is very small, less than 0.1 second.

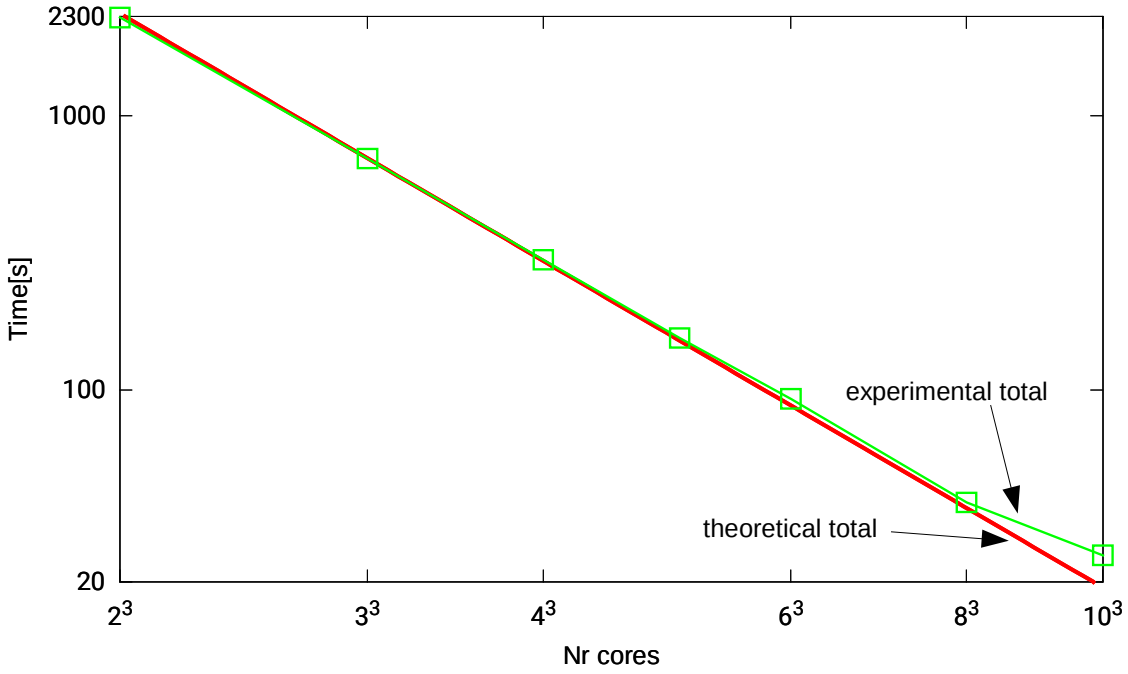


Figure 5.24: Comparison of total experimental and theoretical execution time for $N = 512$ for $p = 3$ for different number of processors $2^3, \ldots, 10^3 = 8, \ldots, 1000$.

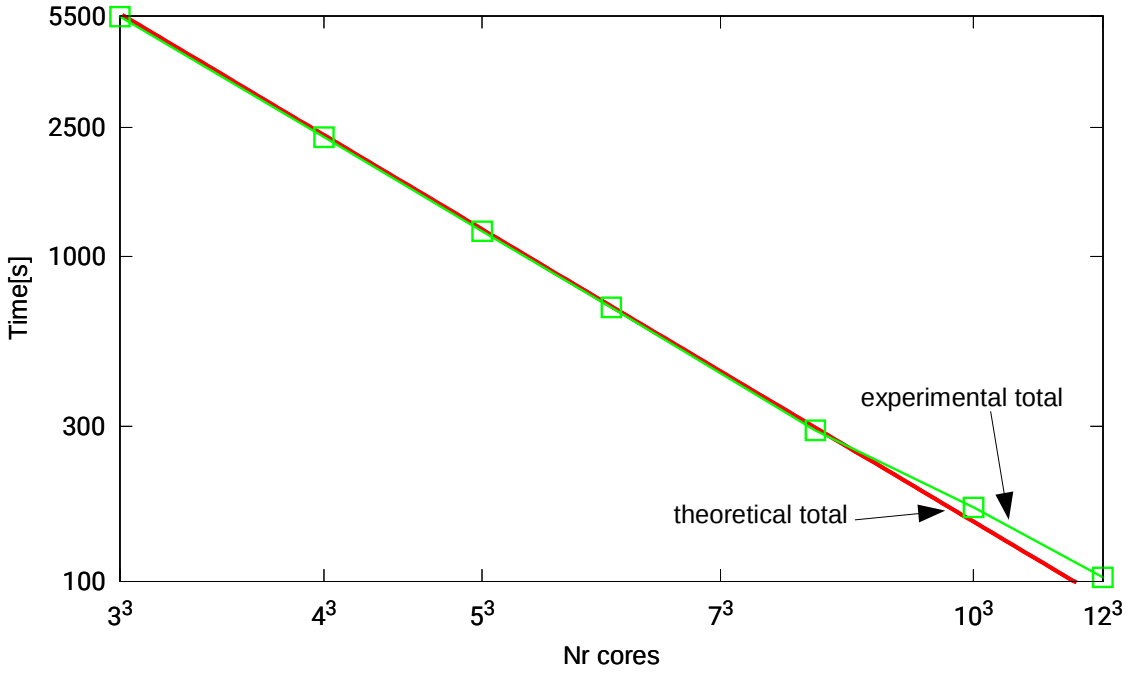


Figure 5.25: Comparison of total experimental and theoretical execution time for $N = 1024$ for $p = 3$ for different number of processors $3^3, \ldots, 12^3 = 27, \ldots, 1728$.

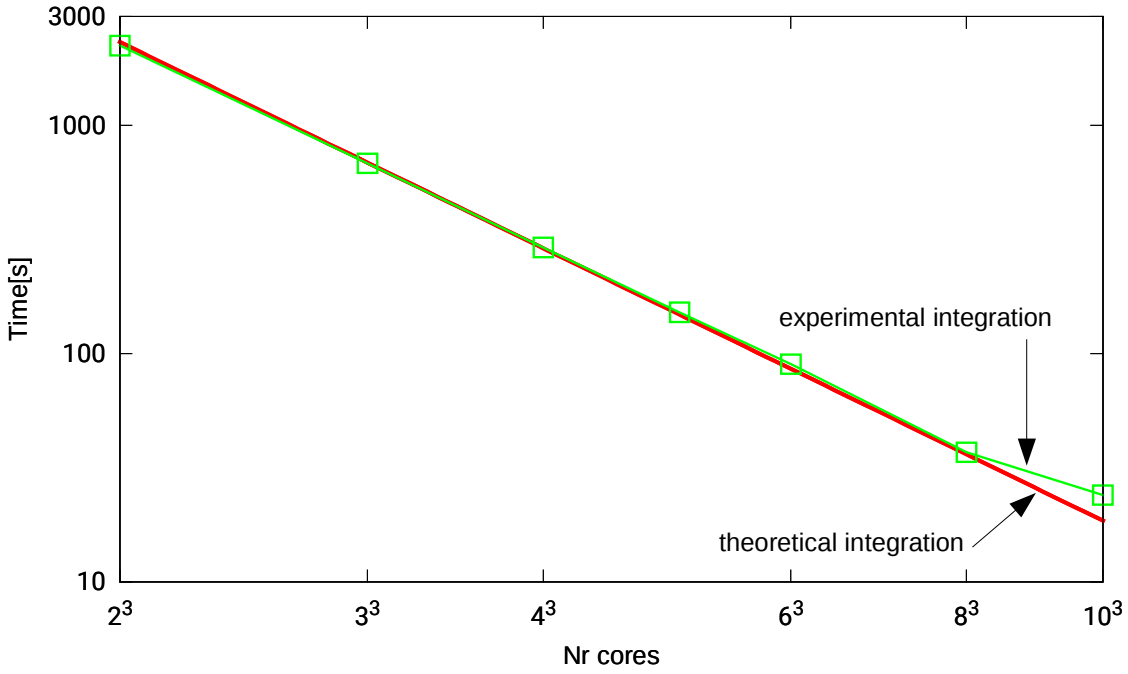


Figure 5.26: Comparison of experimental and theoretical integration time for $N = 512$ for $p = 3$ for different number of processors $2^3, \ldots, 10^3 = 8, \ldots, 1000$.

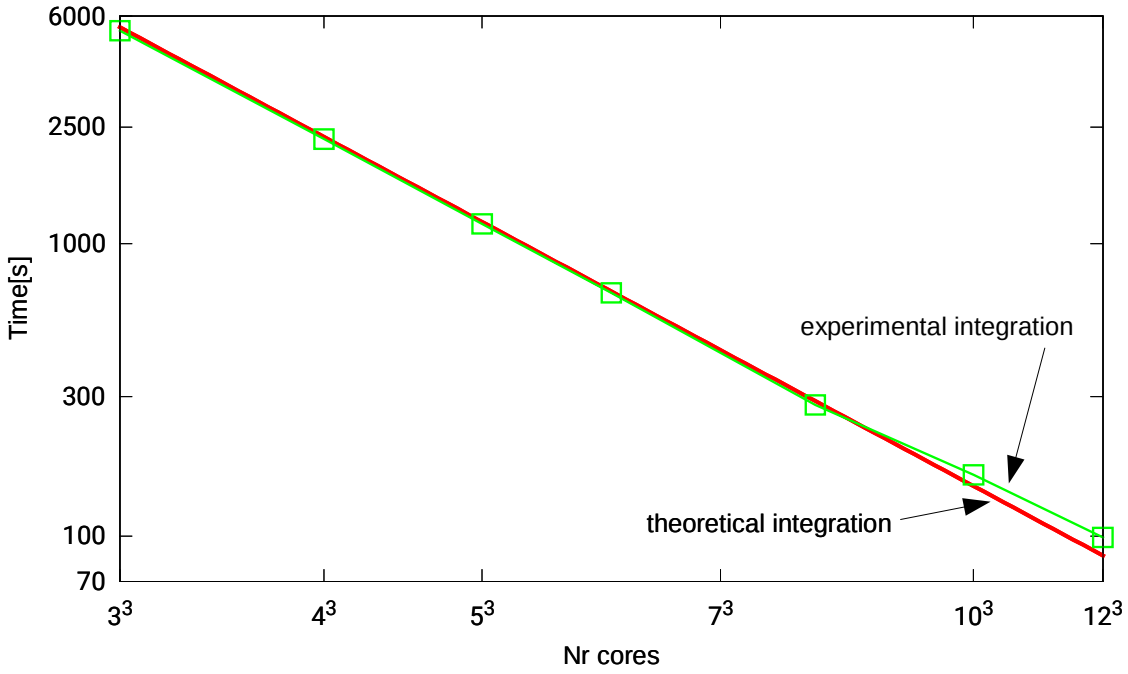


Figure 5.27: Comparison of total experimental and estimated integration time for $N = 1024$ for $p = 3$ for different number of processors $3^3, \ldots, 12^3 = 27, \ldots, 1728$.

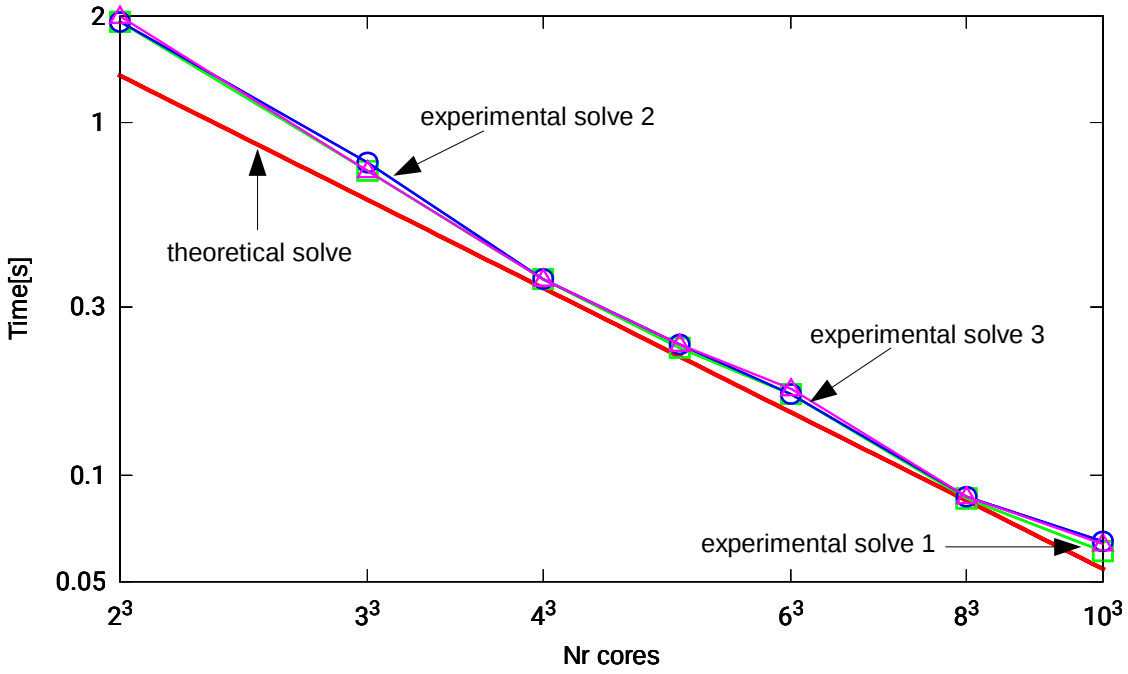


Figure 5.28: Comparison of experimental and theoretical solution times for $N = 512$ for $p = 3$ for different number of processors $2^3, \ldots, 10^3 = 8, \ldots, 1000$

.

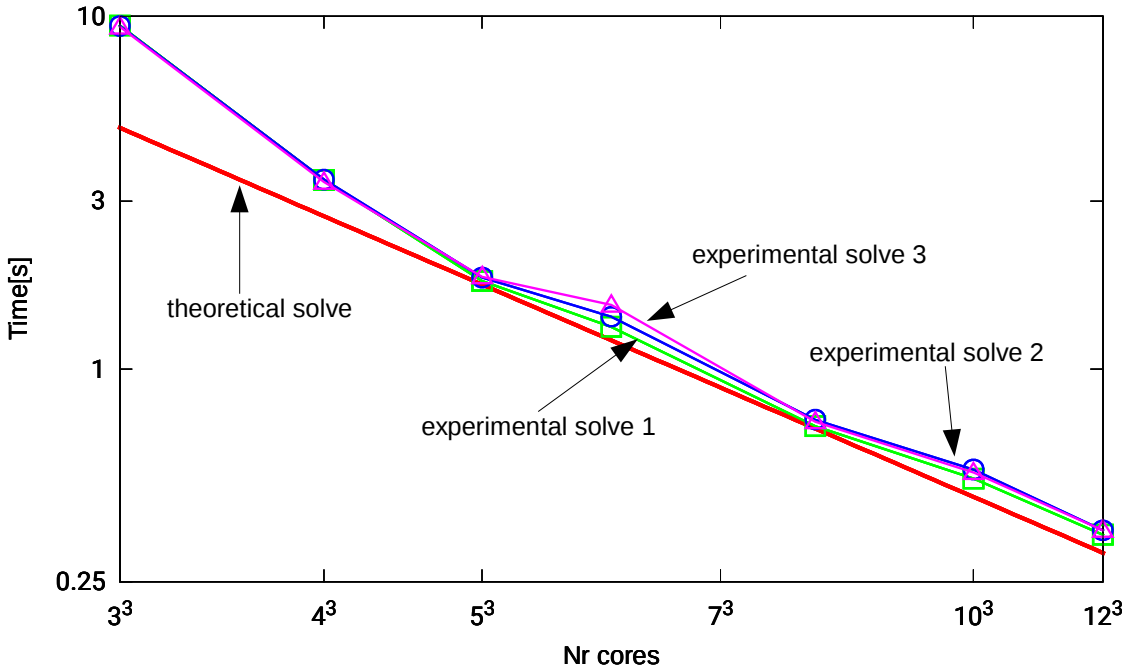


Figure 5.29: Comparison of total experimental and estimated solution times for $N = 1024$ for $p = 3$ for different number of processors $3^3, \ldots, 12^3 = 27, \ldots, 1728$.

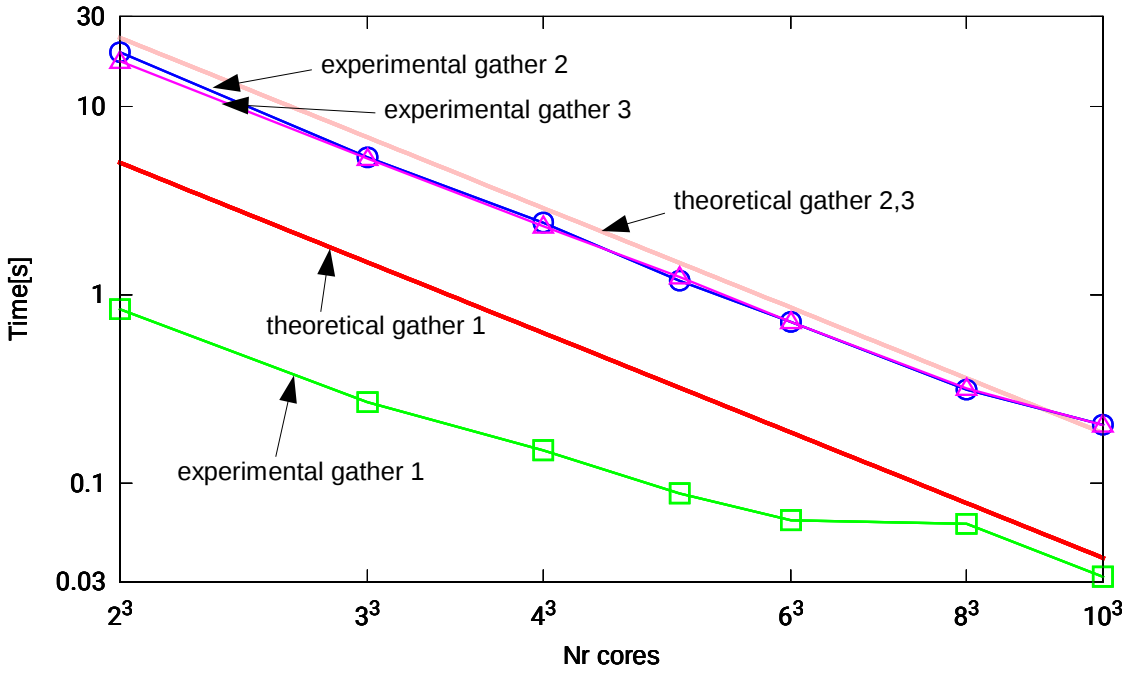


Figure 5.30: Comparison of experimental and theoretical gather times for $N = 512$ for $p = 3$ for different number of processors $2^3, \ldots, 10^3 = 8, \ldots, 1000$.

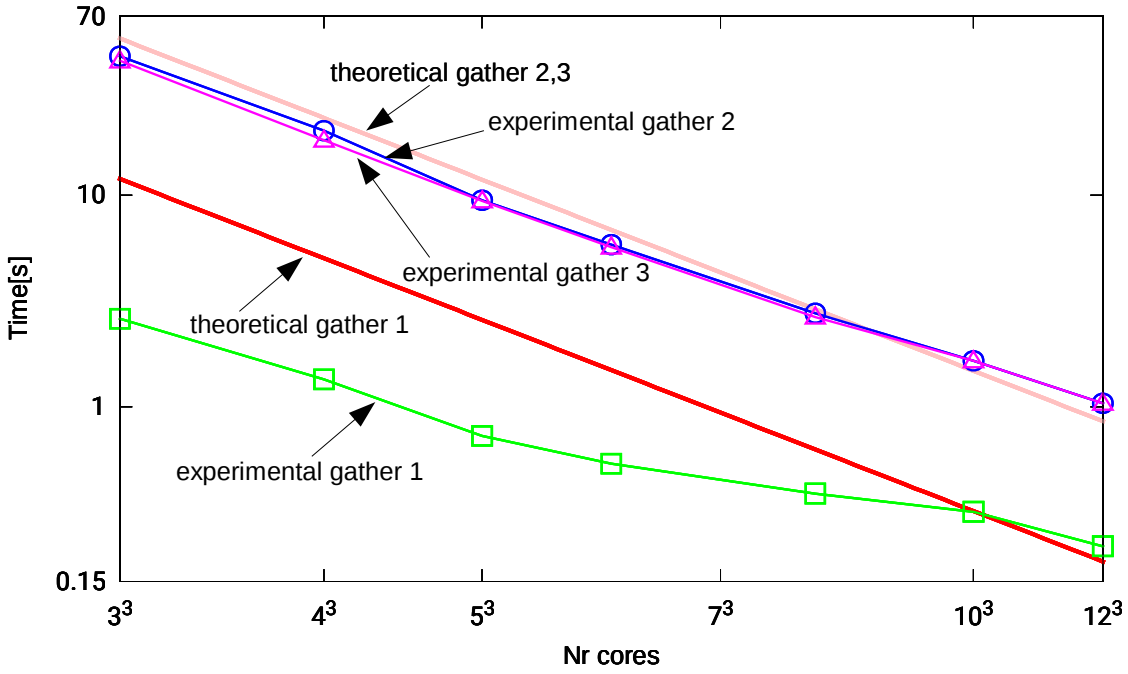


Figure 5.31: Comparison of total experimental and estimated gather times for $N = 1024$ for $p = 3$ for different number of processors $3^3, \ldots, 12^3 = 27, \ldots, 1728$.

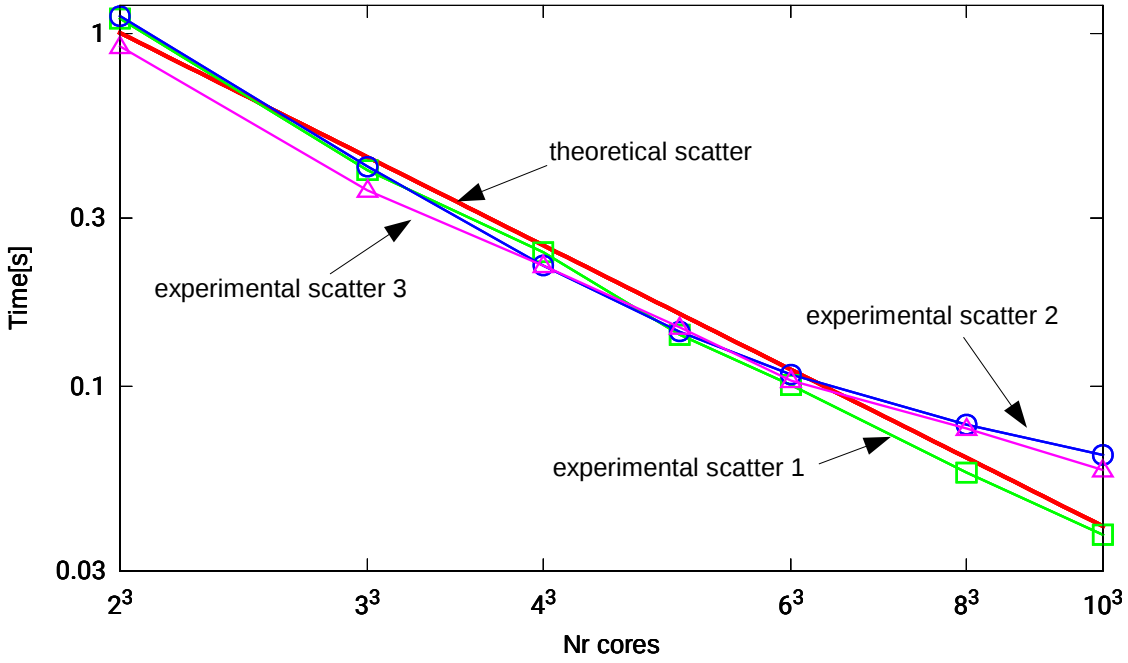


Figure 5.32: Comparison of experimental and theoretical scatter times for $N = 512$ for $p = 3$ for different number of processors $2^3, \ldots, 10^3 = 8, \ldots, 1000$.

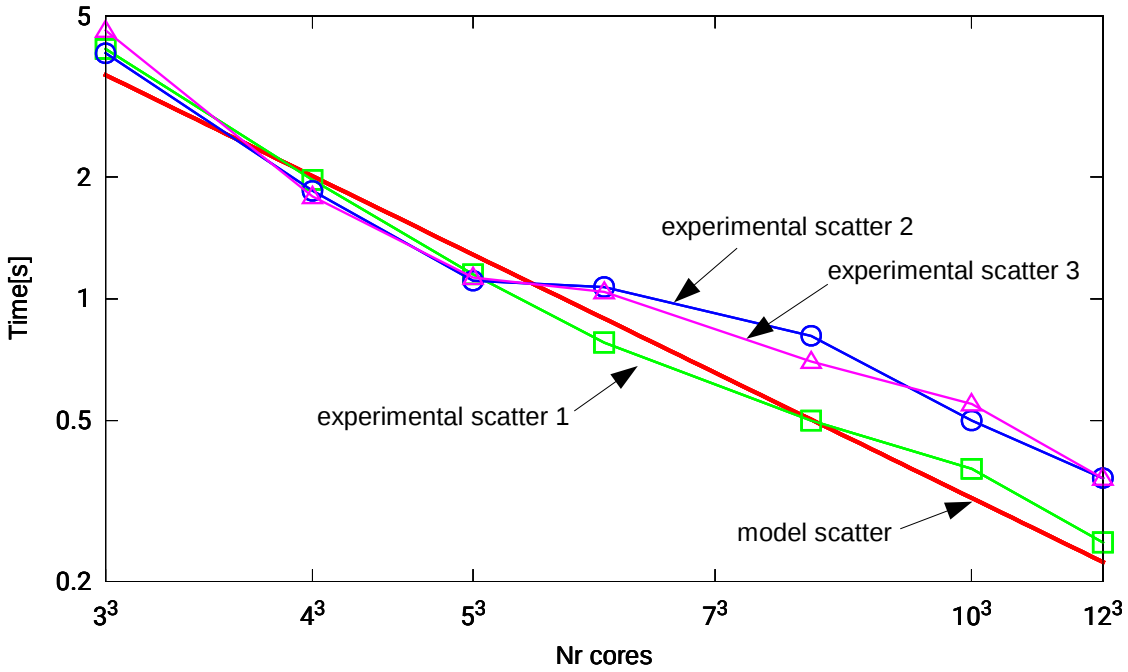


Figure 5.33: Comparison of total experimental and estimated scatter times for $N = 1024$ for $p = 3$ for different number of processors $3^3, \ldots, 12^3 = 8, \ldots, 1728$.

5.3. Integration and solve proportions

This section presents experimental results comparing proportions of integration and solve cost with different solvers for three-dimensional L^2 projection problem. The measurements concern the execution time and FLOPS for sequential integration algorithm and sequential MUPS solver as well as sequential isogeometric L^2 projections. Linear, quadratic, cubic and quartic B-splines were used for comparison.

The numerical results were obtained on SkyFall Linux workstation equpied with Intel Core i7-4820K processor and 64 GB of RAM memory. Both applications were compiled with Intel compiler with highest level of optimization and MKL library. Latest version of MUMPS was used.

The comparison of results for classical approach is presented in Figures 5.34 and 5.35, respectively. Solution part quickly becomes dominant part of total time, while integration is negligible for larger problems. Relatively small problems can be solved in 3D due to memory consuming MUMPS solver.

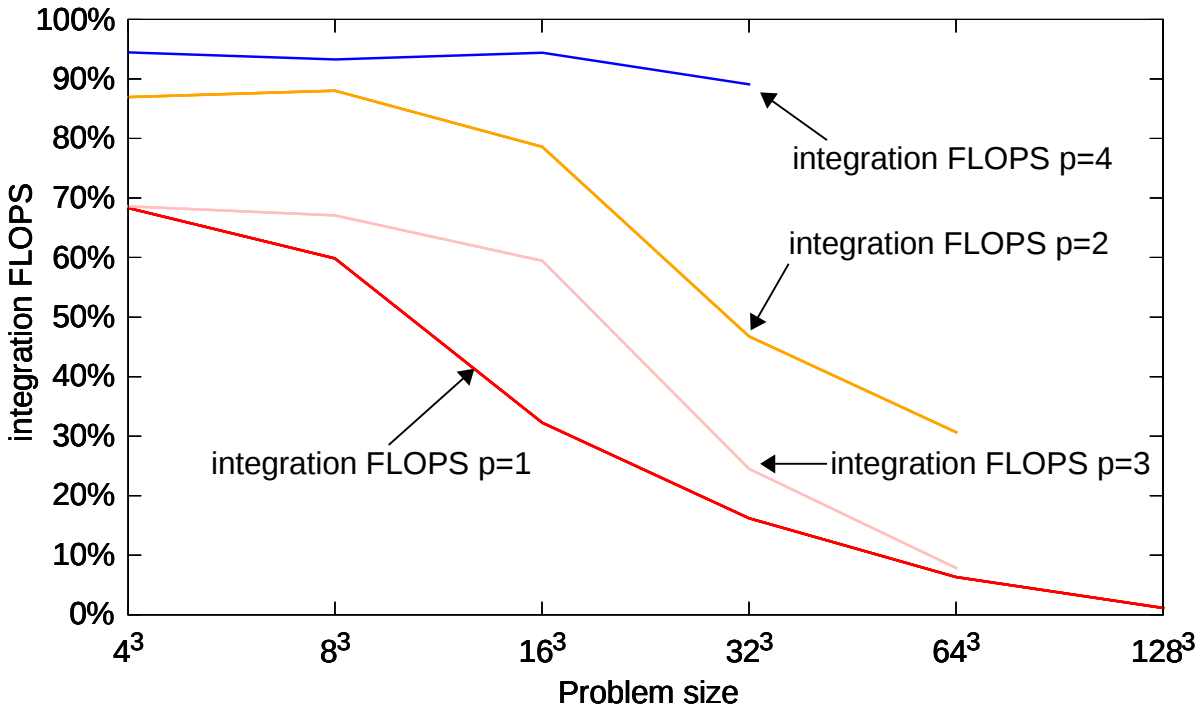


Figure 5.34: Proportion of integration FLOPS to total computation cost with MUMPS solver.

The comparison of results for Alternating Directions Solver is presented in Figures 5.36 and 5.37, respectively. Integration part is dominant part of total time, while solution is negligible cost. Large problems compared to MUMPS can be solved.

As we can easily notice- with Alternating Directions Solver- integration step becomes a bottleneck and should be further accelerated.

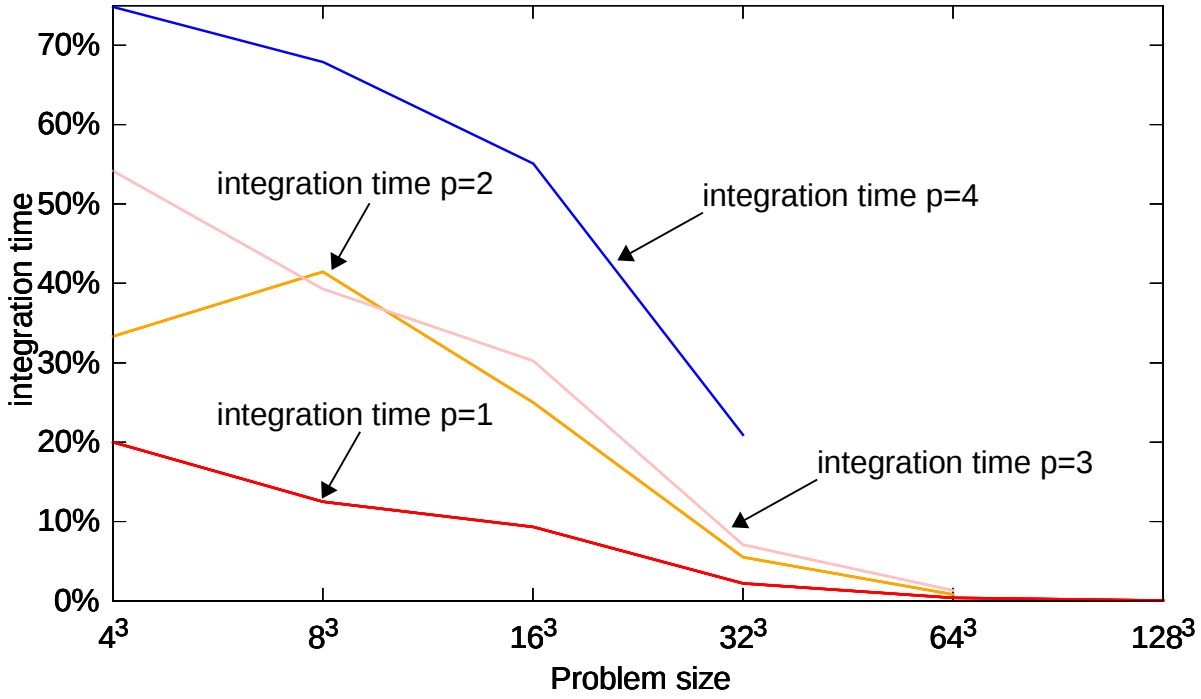


Figure 5.35: Proportion of integration time to total computation cost with MUMPS solver.

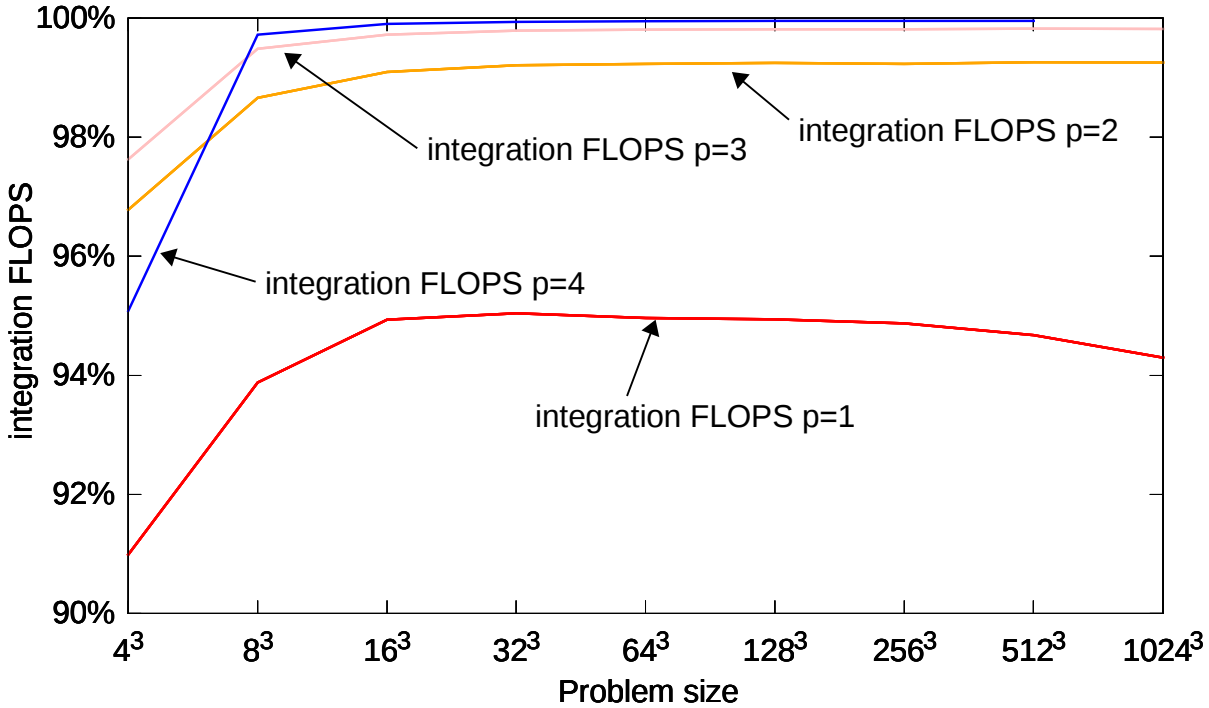


Figure 5.36: Proportion of integration FLOPS to total computation cost with ADS solver.

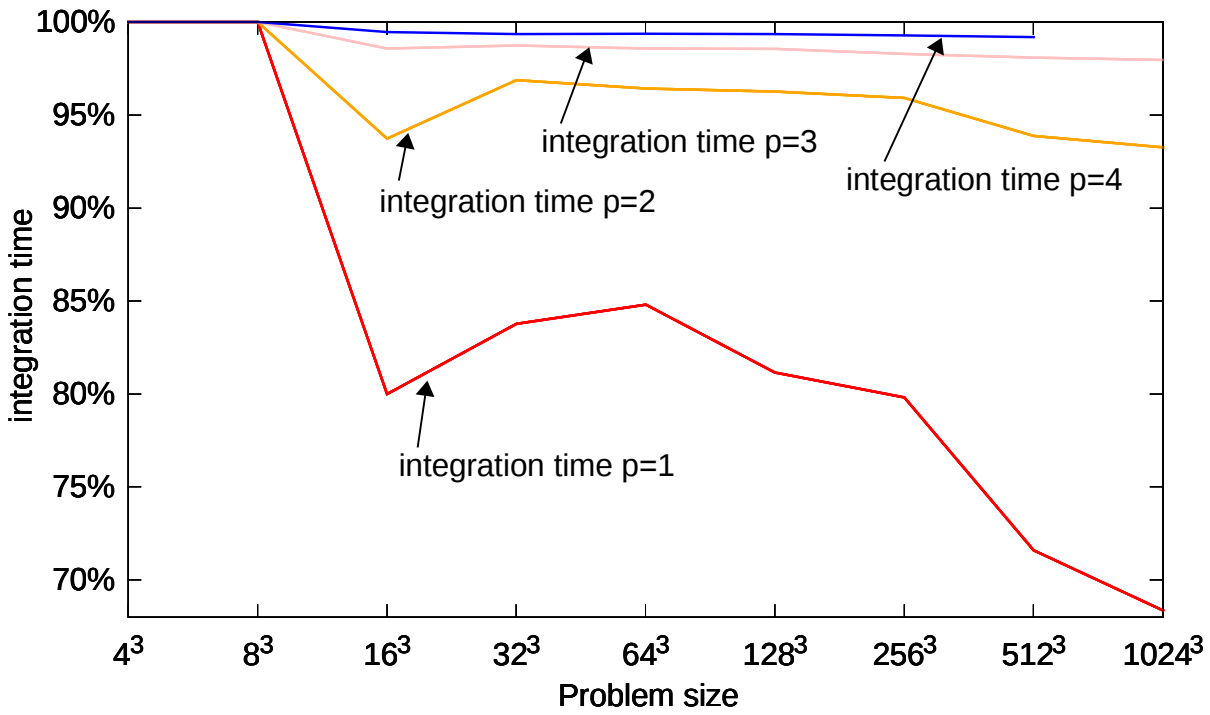


Figure 5.37: Proportion of integration time to total computation cost with ADS solver.

5.4. Comparison of GPU and CPU integration time

This section presents experimental results for linear, quadratic and cubic B-splines, for two-dimensional isogeometric L^2-projection problem. The measurements concern the execution time for sequential integration algorithm executed by CPU and concurrent trace theory based integration algorithm executed over a shared memory GPU.

The numerical results have been obtained on GeForce GTX 780 graphic card equipped with 3 gigabytes of memory and 2304 cores, as well as on NVIDIA Tesla K20c device, which has 5 gigabytes of memory and 2496 cores.

Computations for CPU integration were executed on a SkyFall Linux workstation with Intel Core i7-4820K processor using the latest version of PETIGA compiled with Intel compiler with the highest level of optimization. The CPU computations have been performed using state-of-the-art PETIGA toolkit with efficient implementation of the integration algorithm. Computations for GTX GPU integration were done on a SkyFall Linux workstation with ASUS GTX-780-DC2OC-3GD5 GeForce GTX 780 graphics card compiled with standard nvcc with standard gcc beneath and the highest level of optimization. Computations for Tesla GPU integration were performed on a Merlin Linux server with NVidia Tesla K20c graphics card compiled with standard nvcc with standard gcc beneath and the highest level of optimization.

The comparison of the results for linear, quadratic and cubic B-splines is presented in Figures 5.38, 5.39 and 5.40, respectively.

The concurrent trace theory based integration performed in a shared memory graphic card is between one to two orders of magnitude faster than the integration over a single CPU. When we have more cores available then the problem size, we obtain $\mathcal{O}(logN)$ integration time, compare "GPU integration" in Figures 5.38, 5.39 and 5.40. However, when the problem size grows, we obtain the same linear $\mathcal{O}(N)$ trend for the trace theory-based implementation as for the sequential integration. However we can efficiently

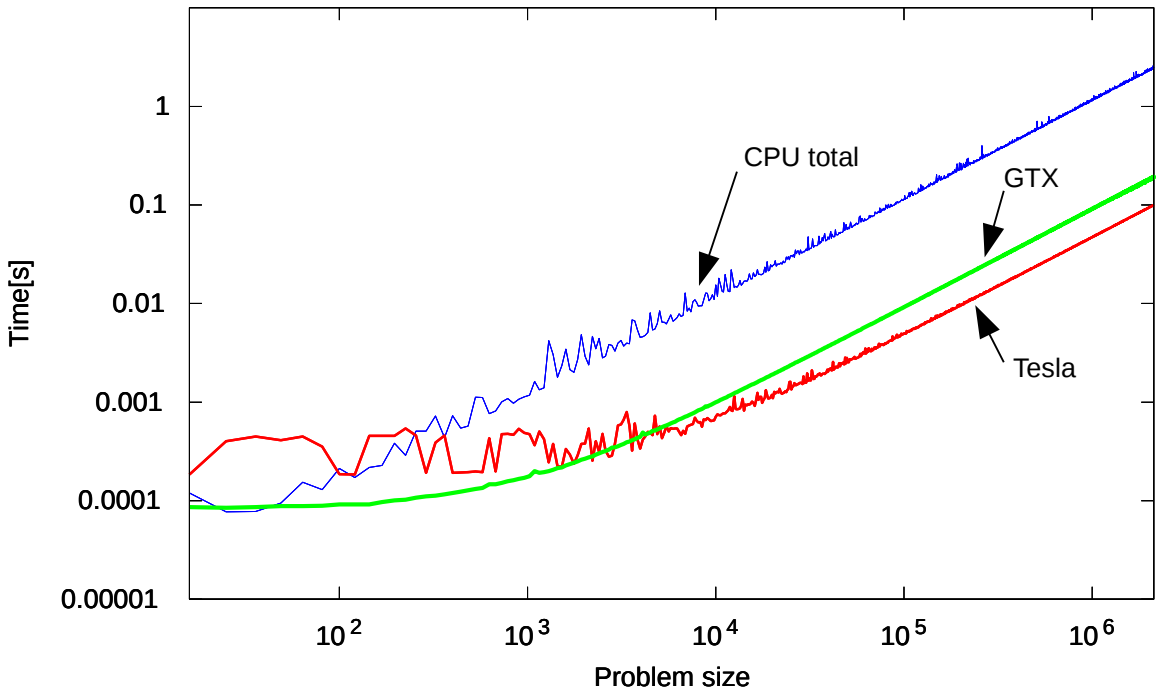


Figure 5.38: Comparison of GPU and CPU integration time for linear B-splines

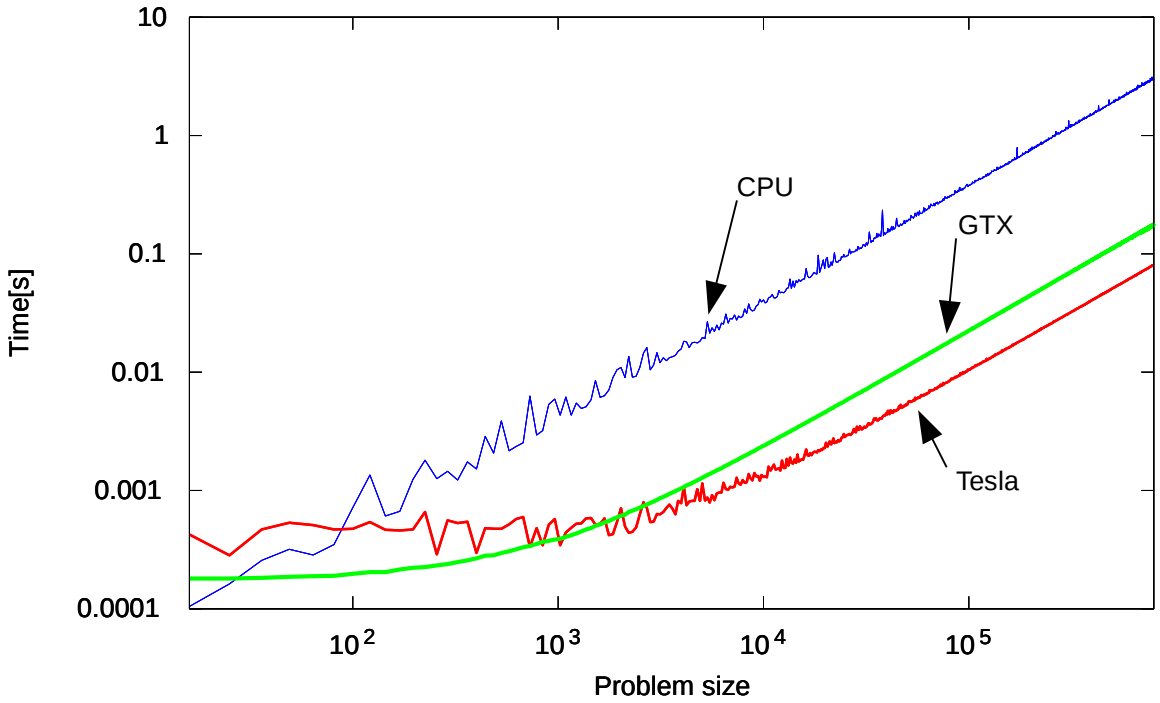


Figure 5.39: Comparison of GPU and CPU integration time for quadratic B-splines

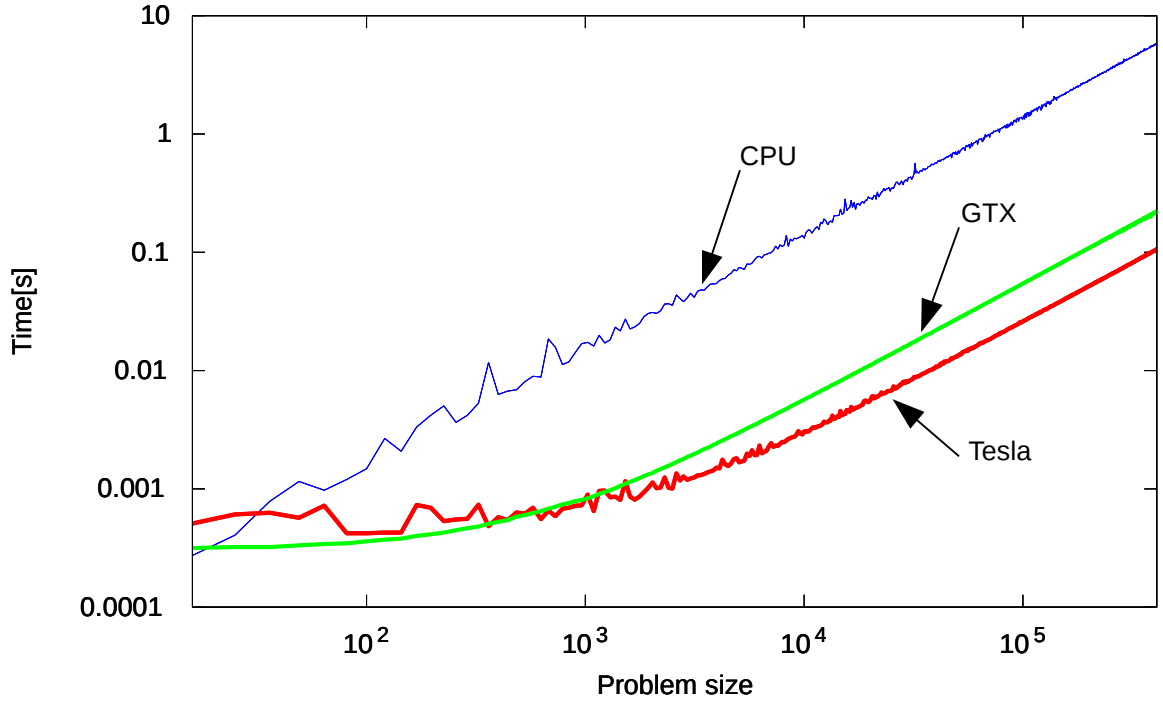


Figure 5.40: Comparison of GPU and CPU integration time for cubic B-splines

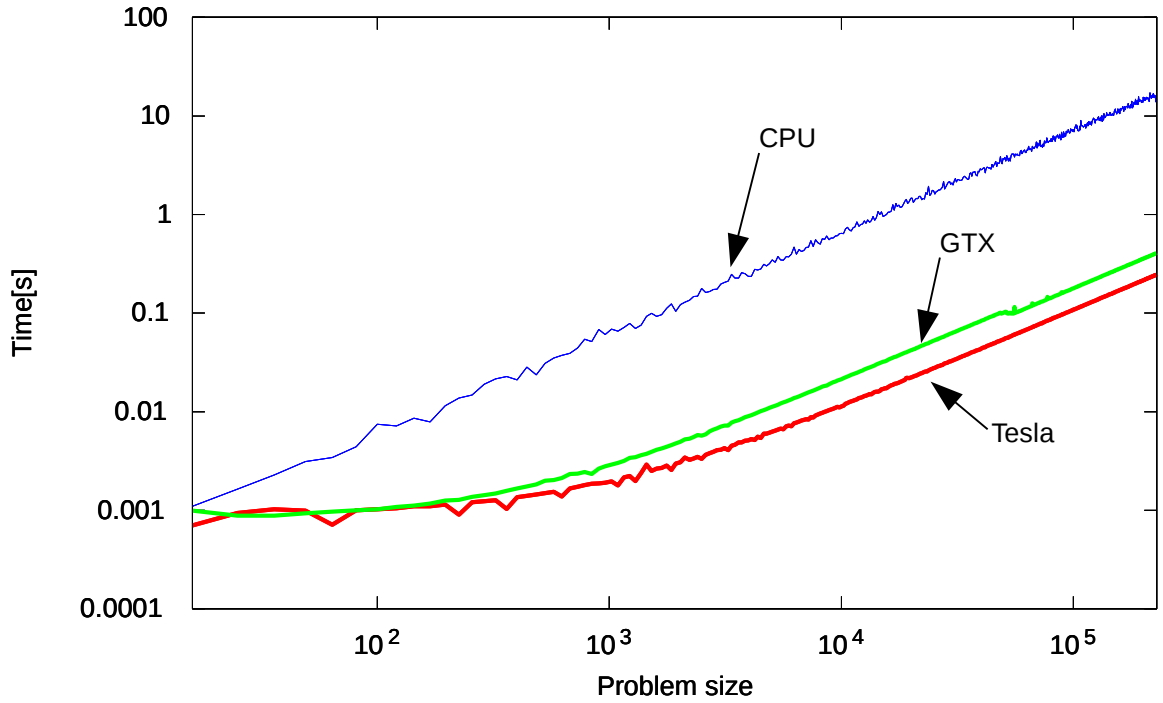


Figure 5.41: Comparison of GPU and CPU integration time for quartic B-splines

utilize all the cores, the constant in front of the cost estimates for the trace theory based implementation is one or two orders of magnitude smaller than in the sequential implementation, compare GTX and Tesla plots in Figures 5.38, 5.39 and 5.40, for two different GPU's, respectively.

5.5. OpenMP

In this section, we present the scalability of the parallel integration using a single Linux cluster node. Namely, the numerical experiments have been performed on the shared-memory node with four Intel R XeonR CPU E7-4860 processors, each possessing ten physical cores (for a total of 40 cores). We utilize quadratic, cubic, and quartic B-splines over a patch of $40 \times 40 \times 40$ finite elements. We report in Figures 5.42-5.44 the total execution time, parallel efficiency and speedup. From the presented experiments in implies that our OpenMP integration scales well up to 15 cores, for quadratic B-splines, up to 20 cores for cubics, for quartics up to 25 cores, and for quintic the efficiency grows up to 35 cores. Higher number of cores results in efficiency below 60 percent.

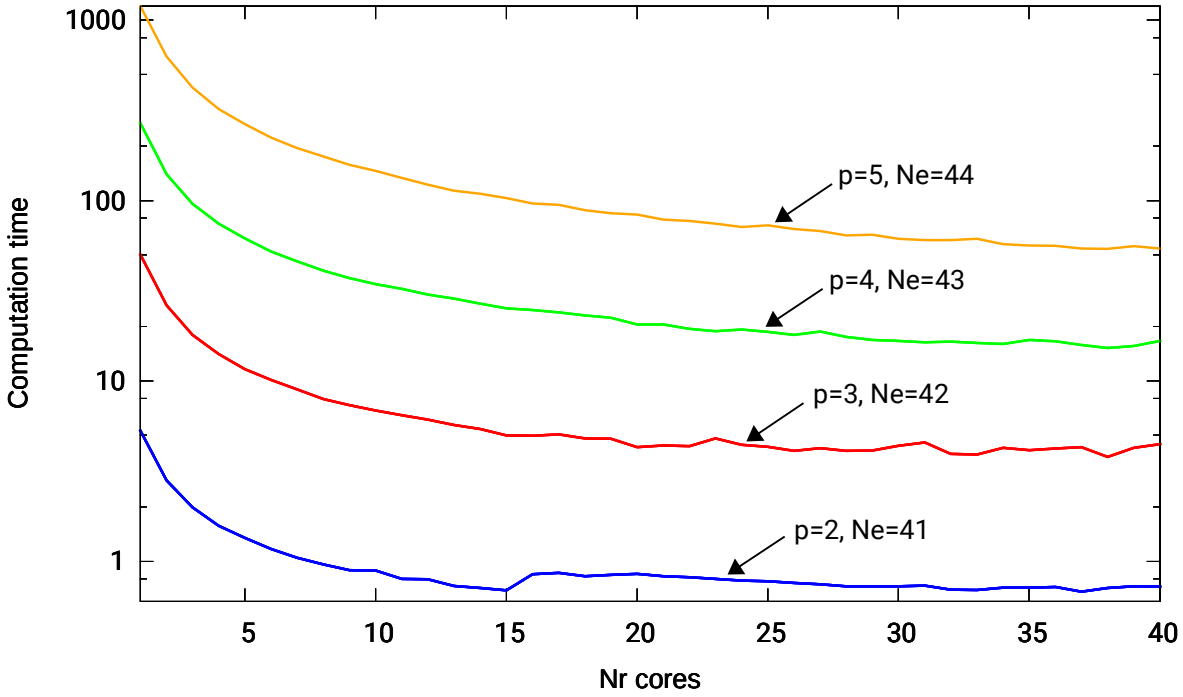


Figure 5.42: Execution time of the parallel integration algorithm in 3D, when increasing number of cores.

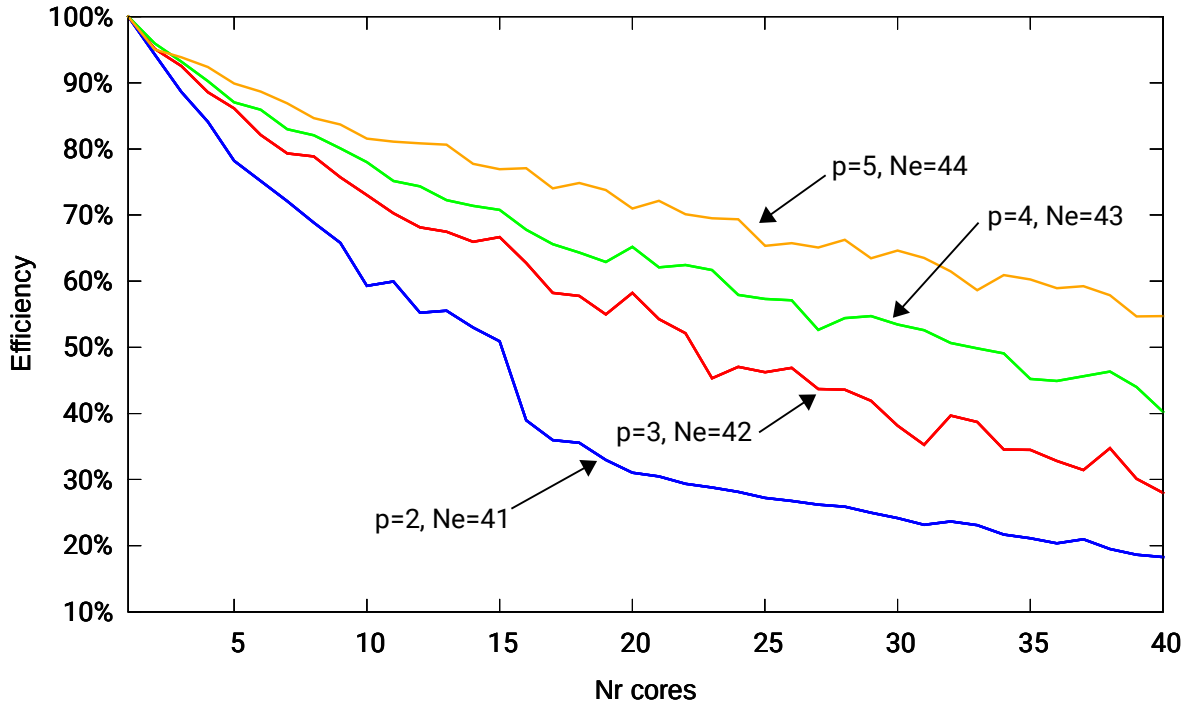


Figure 5.43: Parallel efficiency of the parallel integration algorithm in 3D.

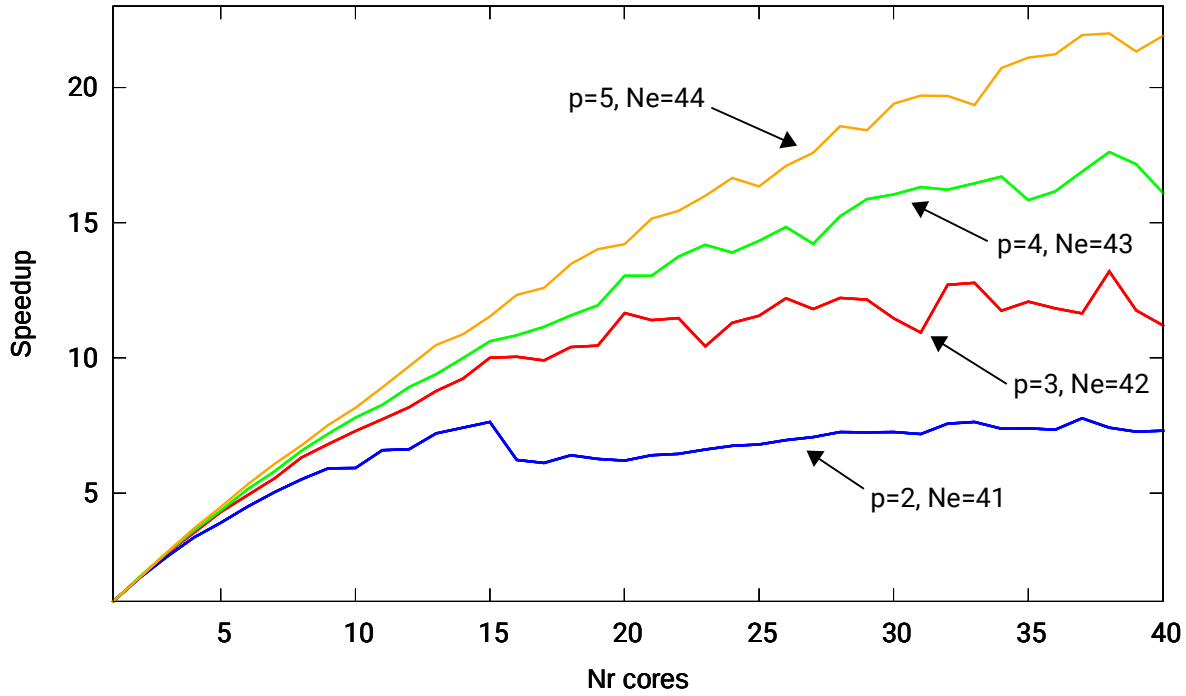


Figure 5.44: Parallel speedup of the parallel integration algorithm in 3D.

5.6. Non-linear flow in heterogenous media

Below we provide results performed with sequential isogeometric L^2 projection algorithm executed over a cube of $n = 16^3$ finite elements with quadratic B-spline basis functions, providing C^1 global continuity of the numerical solutions. We performed 1000 time steps. In Figures 5.45-5.51 we present snapshots of the solution from time steps (size of domain is $[0, 1]^3$ and 160 is the number of layers used by our graphical tool). Additionally, to monitor the stability of the simulation, we present the relative error estimations between simulations executed with two different time step $\Delta t = 10^{-5}$ and $\Delta t = 10^{-6}$. This is dimensionless time step which corresponds to several seconds of real time. This is presented in Figures 5.52. For more details we refer to [75]. The plots present the pressure scalar field obtained from solution of problem of nonlinear flow in heterogenous media (1.12). Namely, we assume that we have a formation layers as presented in Figure 1.1 (oil formation mixed with sand, mud) we pump the fluid to the center of the formation, and observe how the fluid propagates and push out the oil, which can be sucked out by sinks.

We would like to emphasize that all the software and algorithm utilized to obtain this numerical solution has been designed and implemented by us, without any usage of commercial software. The sequential version of our solver needs days to generate such the numerical results (months including debugging and restarting simulations). Our motivation for this work is to reduce the computational cost down to minutes, by utilizing the parallel implementation of the parallel machine providing the best compromise between the execution time and energy consumption.

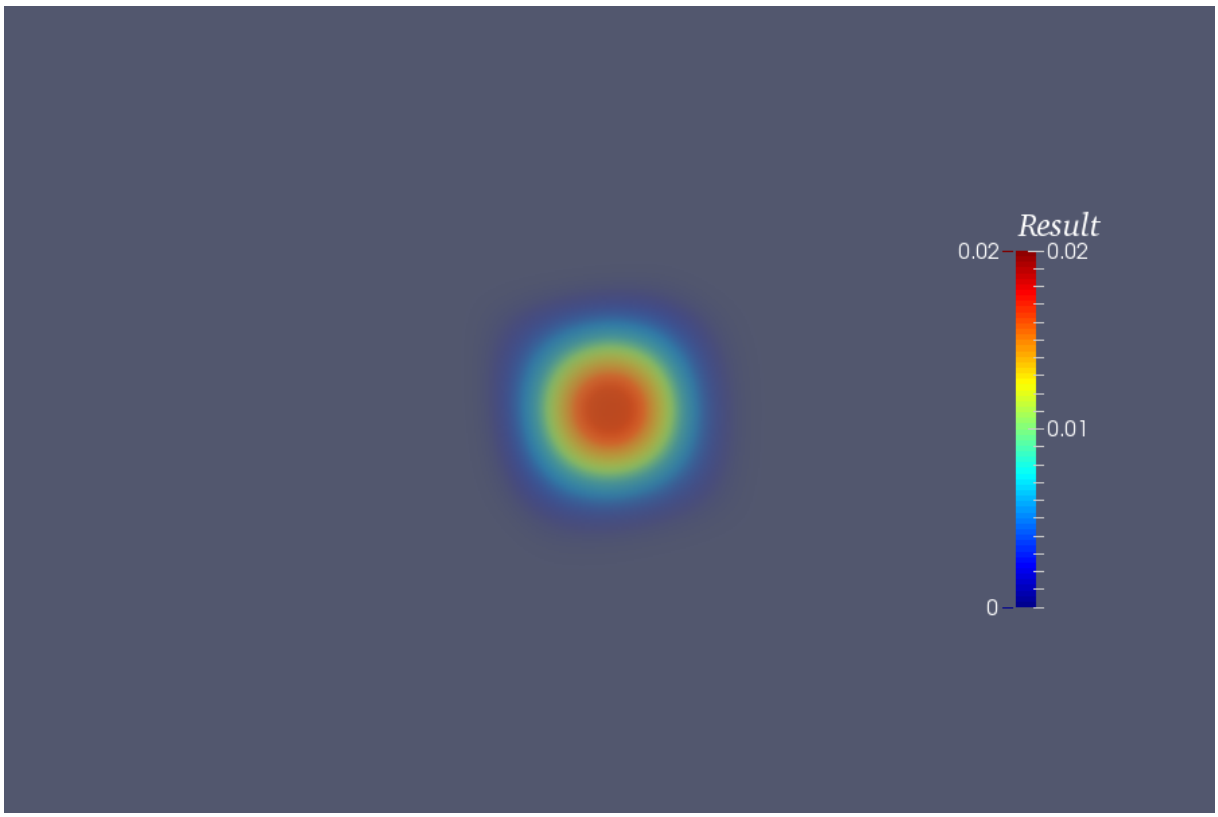


Figure 5.45: Snapshots of the results of the simulations, initial state.

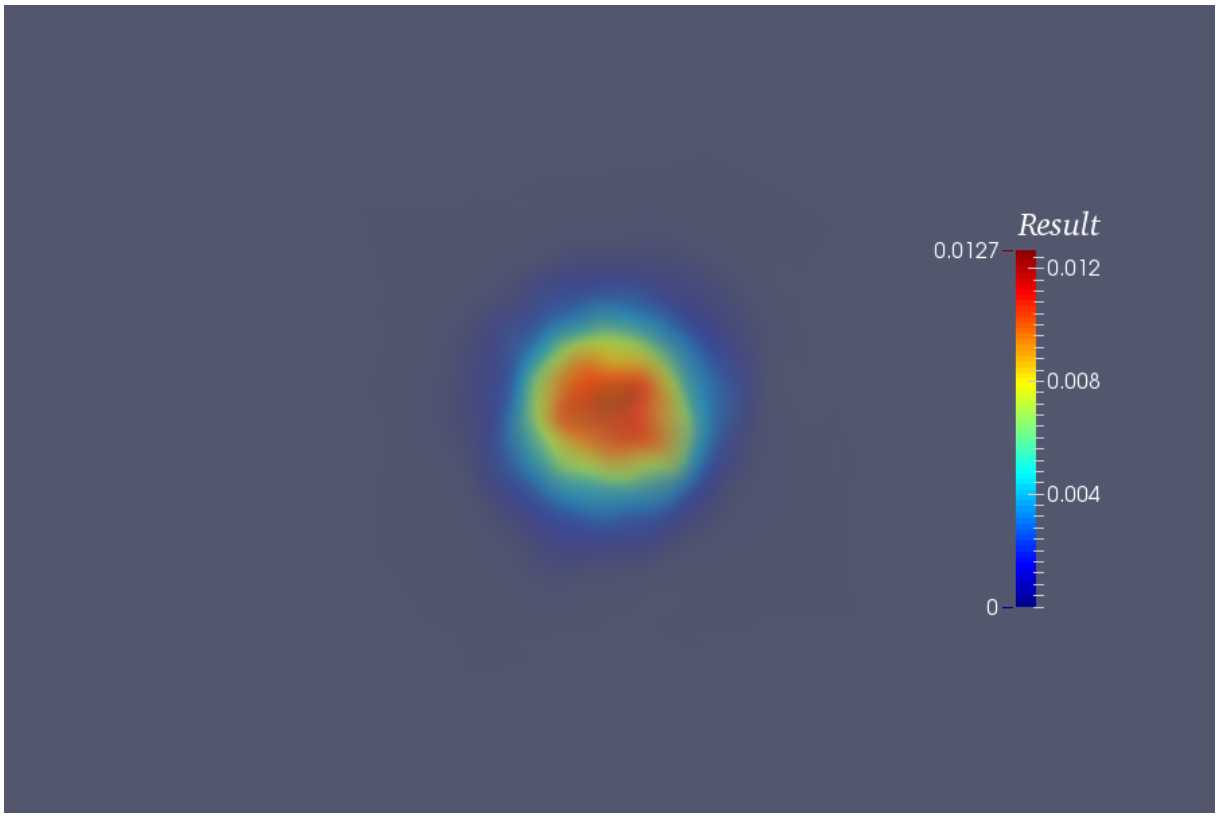


Figure 5.46: Snapshots of the results of the simulations, time step 20.

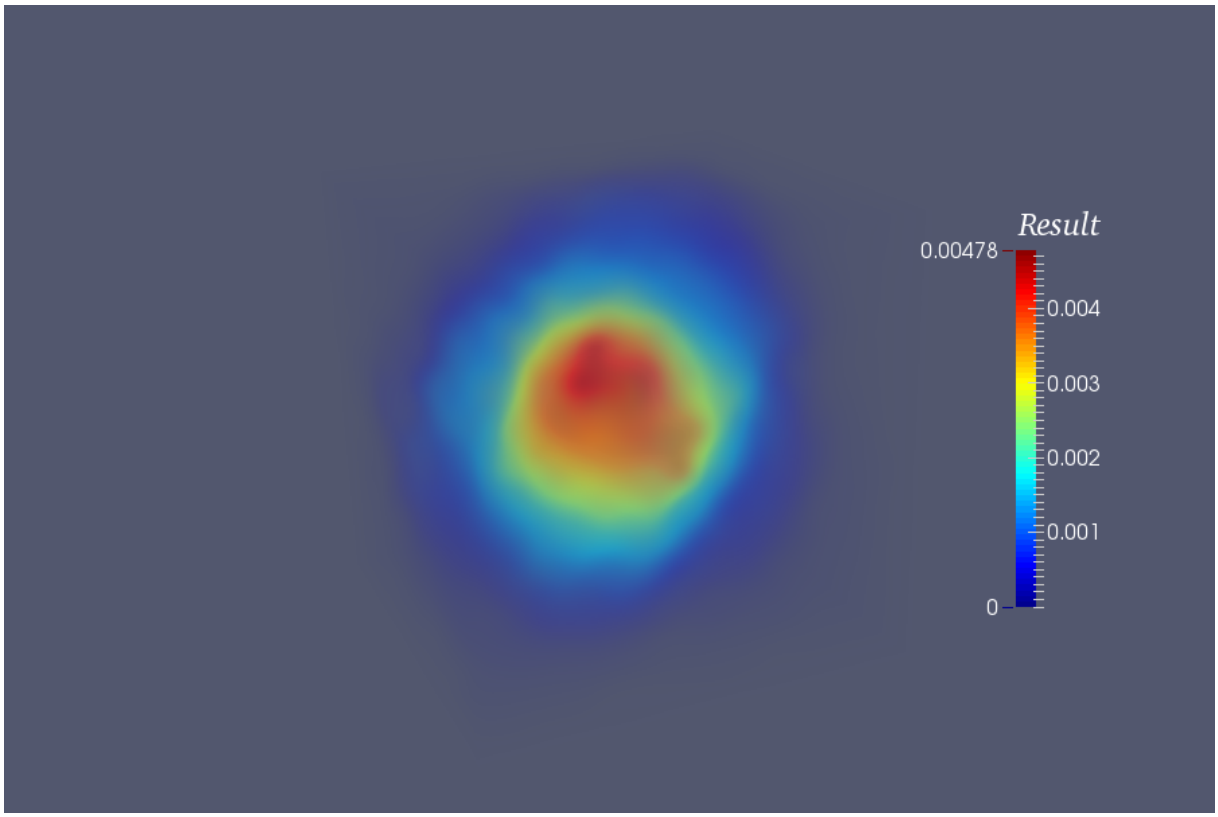


Figure 5.47: Snapshots of the results of the simulations, time step 100.

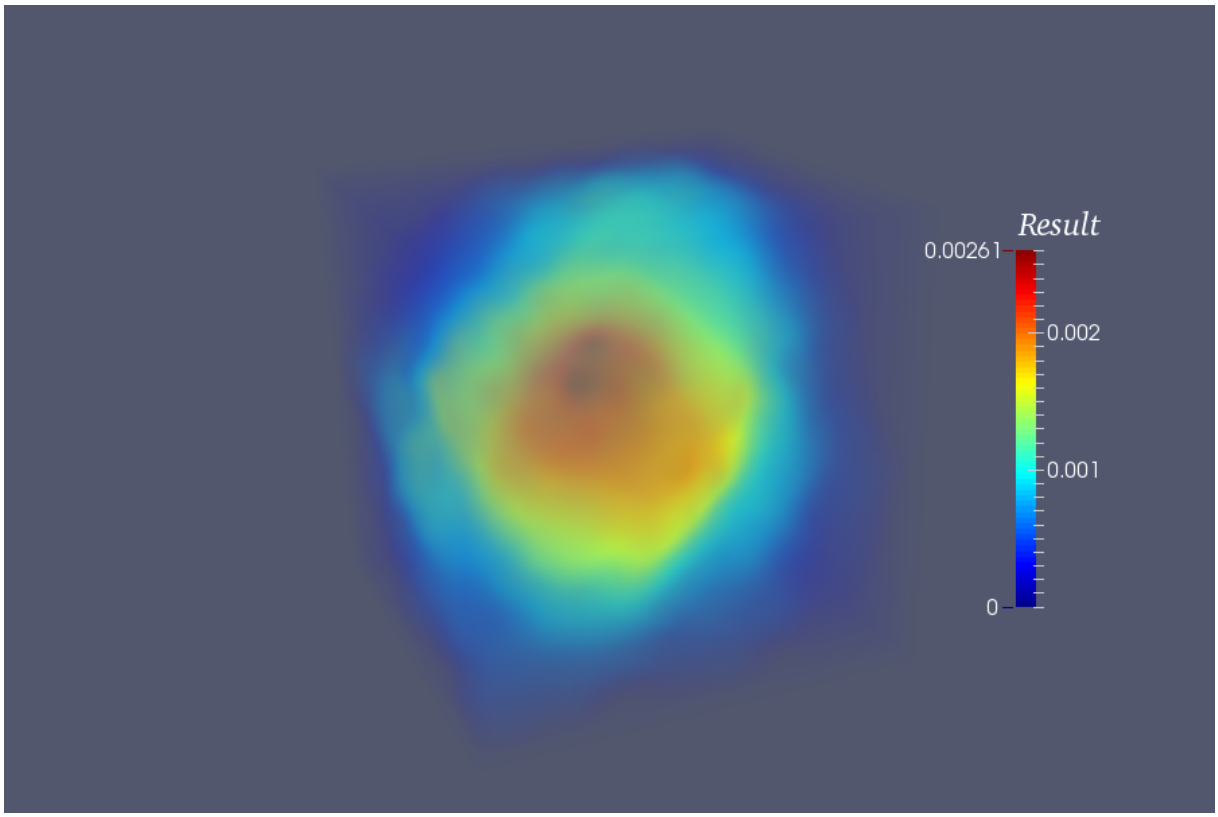


Figure 5.48: Snapshots of the results of the simulations, time step 200.

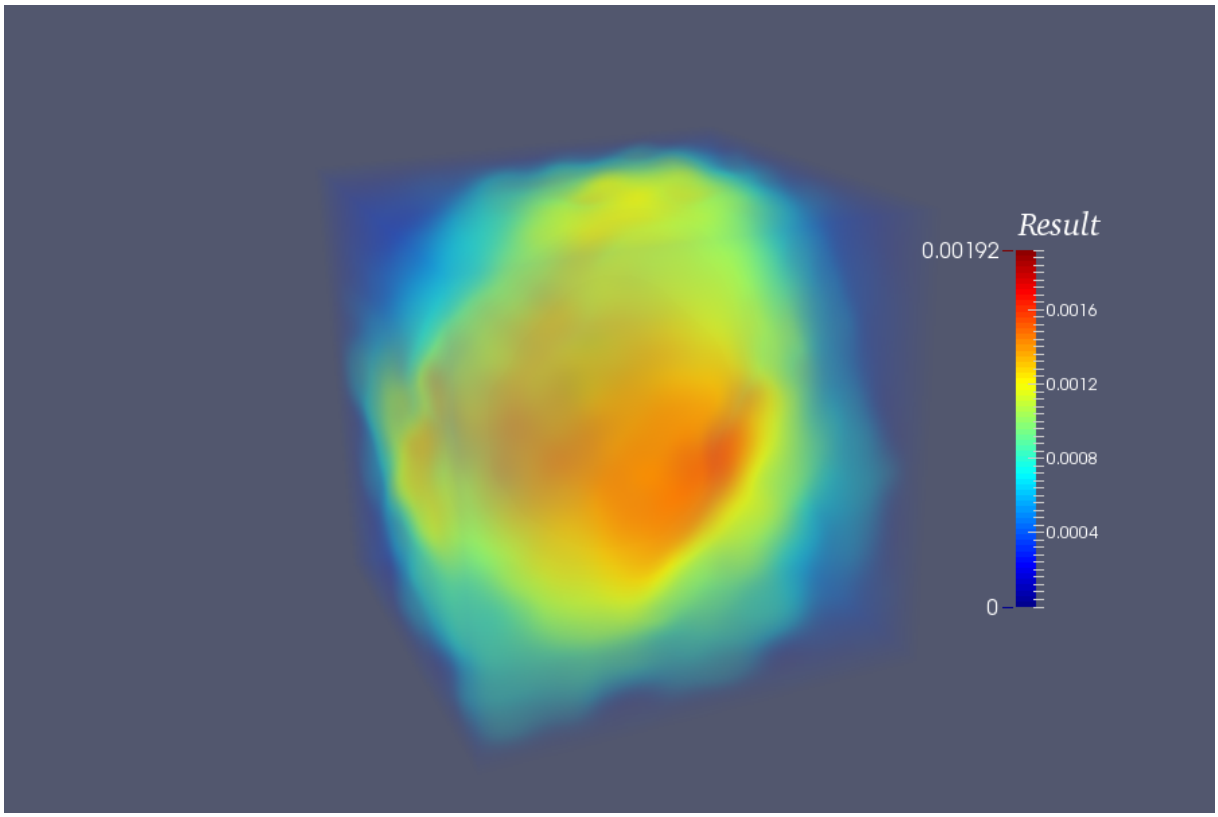


Figure 5.49: Snapshots of the results of the simulations, time step 300.

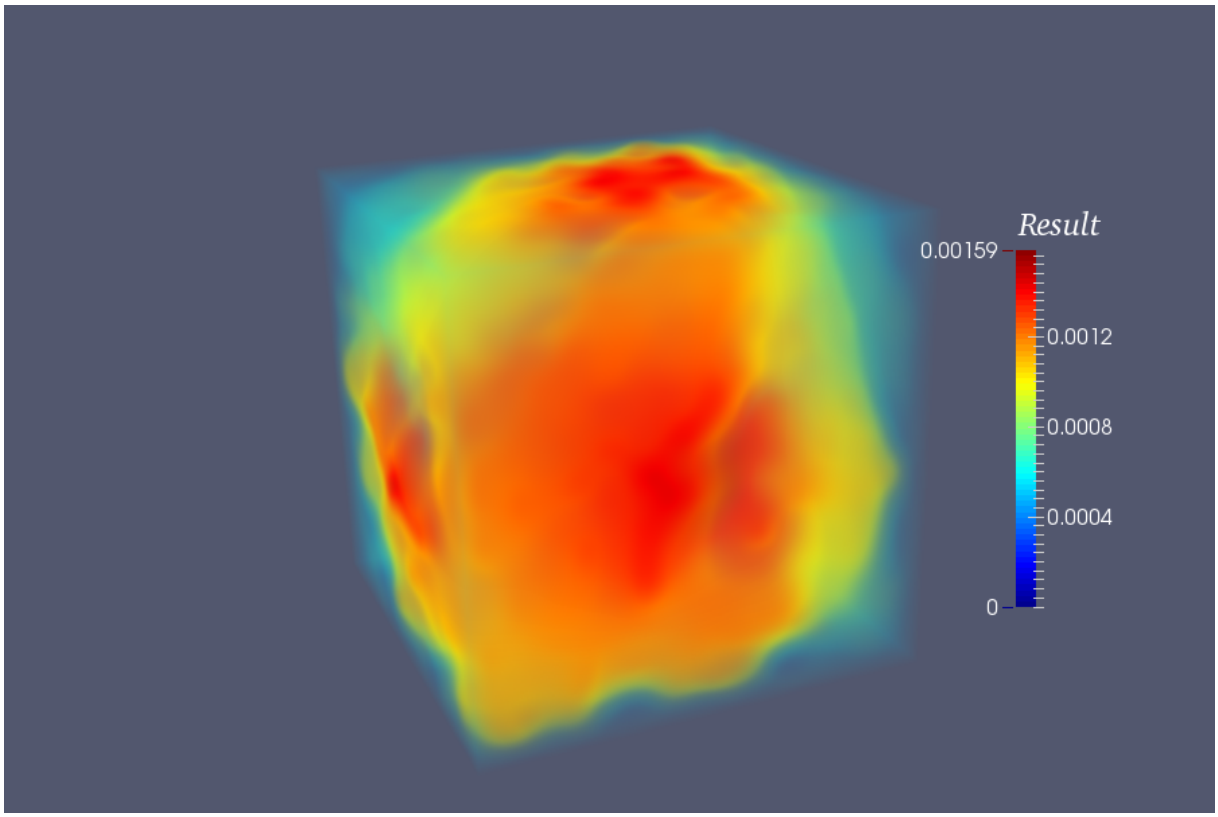


Figure 5.50: Snapshots of the results of the simulations, time step 500.

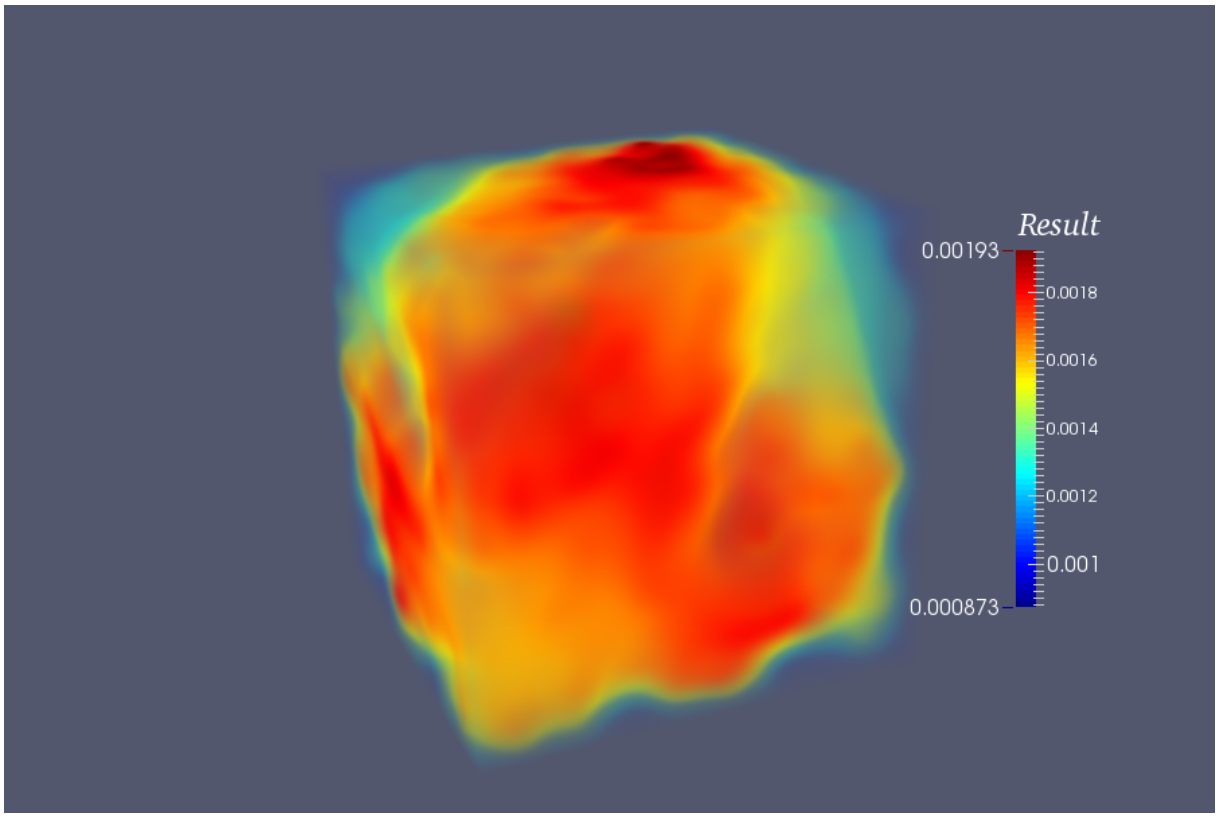


Figure 5.51: Snapshots of the results of the simulations, time step 1000.

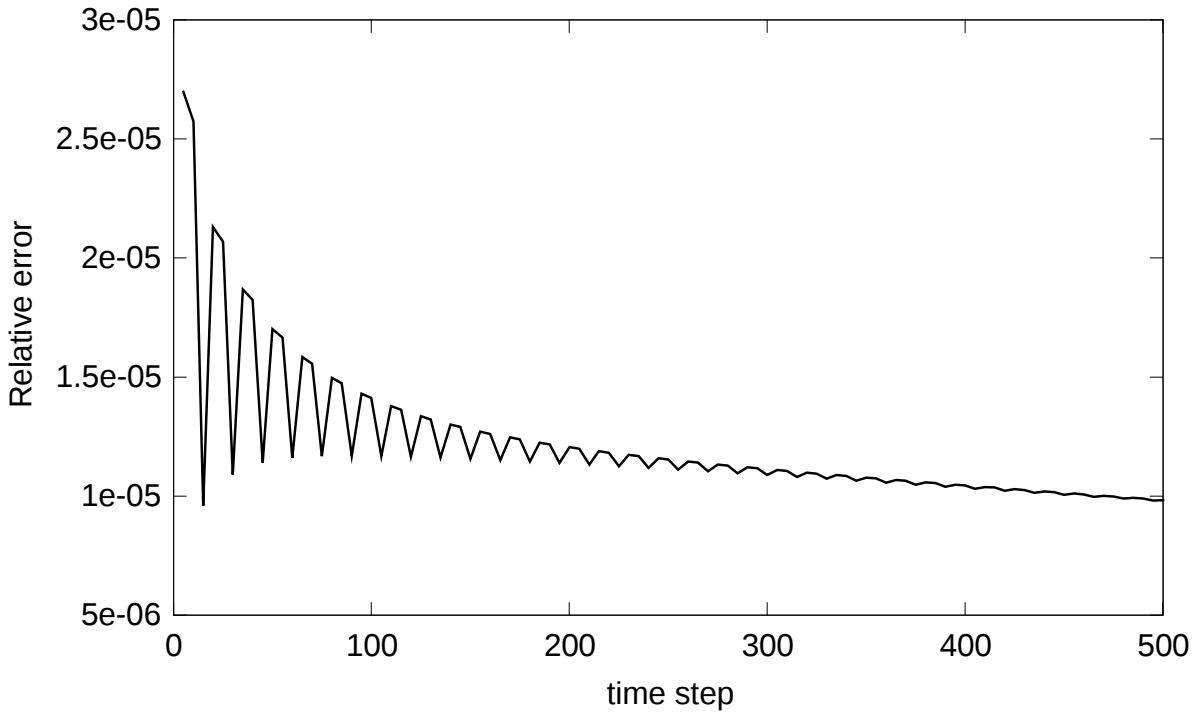


Figure 5.52: Relative error estimation between solutions obtained by different time steps.

Chapter 6

Conclusions and future work

In this dissertation we presented:

- theoretical and experimental analysis of the performance of parallel multi-frontal direct solvers on distributed memory parallel machines
- a fast parallel the alternating directions isogeometric L^2 projections solver
- theoretical and experimental analysis of performance of parallel isogeometric L^2 projection using alternating directions approach
- trace theory based analysis of the parallel integration algorithm with higher order B-spline basis functions

The theoretical estimates assumed sufficiently large number of processors to perform concurrent row subtractions during the local factorizations. We proved 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 showed that the 2D parallel direct solver delivers linear computational and communication costs . In 3D, the computational and communication costs of the parallel solvers grows in terms of the problem size N as $\mathcal{O}(N^{4/3})$ when executed on distributed memory parallel machines. The obtained computational costs estimated for the distributed memory parallel direct solver are similar to those obtained for shared memory parallel machines [79]. The difference in the derivation is that here it appears an additional term related to the communication cost. The theoretical estimates were verified with numerical experiments.

Our parallel alternating direction solver will be available through PETIGA library [18]. The computational complexity of the parallel algorithm is of the order $\mathcal{O}\left(\frac{p^6 N}{c}\right)$ and the communication complexity is of the order $\mathcal{O}\left(\frac{N}{c^{2/3}}\right)$, where p denotes the order of the B-spline basis with C^{p-1} global continuity, N denotes the number of elements and c the number of processors over the 3D hypercube. The theoretical estimates were verified by numerical experiments performed over the LONESTAR linux cluster from Texas Advanced Computing Center. In particular we showed that we can solve the 3D isogeometric L^2 projection problem with $512^3 = 134,217,728$ unknowns within 20 seconds and $1024^3 = 1,073,741,824 = \mathcal{O}(10^9)$ unknowns within 3 minutes by using 1000 processors from the LONESTAR Linux cluster. We are not aware of any other solver delivering such fast solution for $100 - 1000$ millions of unknowns.

The sequential alternating direction isogeometric solver was applied for solution of non-stationary problem of nonlinear flows in highly-heterogeneous porous media. The solver performed 1000 time steps in order to simulate the flow through the entire domain. The correctness of the solution was verified by checking the relative error for two simulations with different time steps as well as by controlling the energy of the solution in particular time steps.

We provided sets of tasks that can be automatically scheduled and executed concurrently, set by set, on GPU. The concurrent integration algorithm executed on CPU was compared with the integration algorithm execution on GPU. The algorithms were tested on the exemplary two-dimensional L^2-projection problems, however, the construction of the concurrent algorithm remains the same if we replace the L^2-projection problem with any other two-dimensional elliptic problem. The concurrent integration executed on GPU is between one and two orders of magnitude faster than sequential integration performed on CPU, for two-dimensional problems up to one million degrees of freedom.

The future work may involve replacement of c^2 sequential solves over a face of the 3D cube by c^2 parallel solves executed within rows of a cube of processors, utilizing the parallel multi-frontal one dimensional isogeometric solver [57]. This however will not affect the general scalability of the solver, since at this point the integration time is dominating the entire solution. An alternative way of improvement of the solver scalability would be to consider some fast integration schemes for B-spline basis functions. It may also include generalization of the method to non-uniform adapted grids with T-splines technique [36]. We also consider expression of the alternating direction algorithm by graph grammar productions and Petri nets, as it has been done for two and three-dimensional finite element method [62, 63, 74, 75]. Concurrent integration algorithm on GPU can be extended to 3D case.

Appendices

Appendix A

Alternating directions for isogeometric L^2 projections

Following [45, 46] we describe the L^2 alternating direction solver that reduces the two-dimensional or three-dimensional L^2 projection problem into 2 or 3 1D problems with multiple right-hand sides. The projection problem can be summarized as $min||\sum_{i=1} b_i B_i - f||_{l^2}$ which is equivalent to

$$\begin{pmatrix} \begin{bmatrix} \int_\Omega (B_1^y B_1^x)(B_1^y B_1^x) & \dots & \int_\Omega (B_1^y B_1^x)(B_1^y B_{N_x}^x) \\ \vdots & \ddots & \vdots \\ \int_\Omega (B_1^y B_{N_x}^x)(B_1^y B_1^x) & \dots & \int_\Omega (B_1^y B_{N_x}^x)(B_1^y B_{N_x}^x) \end{bmatrix} & \dots & \begin{bmatrix} \int_\Omega (B_1^y B_1^x)(B_{N_y}^y B_1^x) & \dots & \int_\Omega (B_1^y B_1^x)(B_{N_y}^y B_{N_x}^x) \\ \vdots & \ddots & \vdots \\ \int_\Omega (B_1^y B_{N_x}^x)(B_{N_y}^y B_1^x) & \dots & \int_\Omega (B_1^y B_{N_x}^x)(B_{N_y}^y B_{N_x}^x) \end{bmatrix} \\ \vdots & \ddots & \vdots \\ \begin{bmatrix} \int_\Omega (B_{N_y}^y B_1^x)(B_{N_y}^y B_1^x) & \dots & \int_\Omega (B_{N_y}^y B_1^x)(B_1^y B_{N_x}^x) \\ \vdots & \ddots & \vdots \\ \int_\Omega (B_{N_y}^y B_{N_x}^x)(B_{N_y}^y B_1^x) & \dots & \int_\Omega (B_{N_y}^y B_{N_x}^x)(B_1^y B_{N_x}^x) \end{bmatrix} & \dots & \begin{bmatrix} \int_\Omega (B_{N_y}^y B_1^x)(B_{N_y}^y B_1^x) & \dots & \int_\Omega (B_{N_y}^y B_1^x)(B_{N_y}^y B_{N_x}^x) \\ \vdots & \ddots & \vdots \\ \int_\Omega (B_{N_y}^y B_{N_x}^x)(B_{N_y}^y B_1^x) & \dots & \int_\Omega (B_{N_y}^y B_{N_x}^x)(B_{N_y}^y B_{N_x}^x) \end{bmatrix} \end{pmatrix} \begin{pmatrix} \begin{bmatrix} b_{1,1} \\ \vdots \\ b_{1,N_x} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} b_{N_y,1} \\ \vdots \\ b_{N_y,N_x} \end{bmatrix} \end{pmatrix} = \begin{pmatrix} \begin{bmatrix} \int_\Omega (B_1^y B_1^x) f \\ \vdots \\ \int_\Omega (B_1^y B_{N_x}^x) f \end{bmatrix} \\ \vdots \\ \begin{bmatrix} \int_\Omega (B_{N_y}^y B_1^x) f \\ \vdots \\ \int_\Omega (B_{N_y}^y B_{N_x}^x) f \end{bmatrix} \end{pmatrix} \tag{A.1}$$

the two-dimensional B-spline bases are built from tensor products of two one dimensional bases $B^x = \{B_1^x, \dots, B_{N_x}^x\}$ and $B^y = \{B_1^y, \dots, B_{N_y}^y\}$.

We notice that from the Fubini iterative integration method

$$\int_{\Omega} g_1(x)g_2(y) = \int_x \int_y g_1(x)g_2(y) = \int_x g_1(x) \int_y g_2(y) \tag{A.2}$$

we have

$$\left(\begin{array}{ccc} \begin{bmatrix} \int_y (B_1^y B_1^y) \int_x (B_1^x B_1^x) & \dots & \int_y (B_1^y B_1^y) \int_x (B_1^x B_{N_x}^x) \\ \vdots & \ddots & \vdots \\ \int_y (B_1^y B_1^y) \int_x (B_{n_x}^x B_1^x) & \dots & \int_y (B_1^y B_1^y) \int_x (B_{N_x}^x B_{N_x}^x) \end{bmatrix} & \dots & \begin{bmatrix} \int_y (B_1^y B_{N_y}^y) \int_x (B_1^x B_1^x) & \dots & \int_y (B_1^y B_{N_y}^y) \int_x (B_1^x B_{N_x}^x) \\ \vdots & \ddots & \vdots \\ \int_y (B_1^y B_{N_y}^y) \int_x (B_{N_x}^x B_1^x) & \dots & \int_y (B_1^y B_{N_y}^y) \int_x (B_{N_x}^x B_{N_x}^x) \end{bmatrix} \\ \vdots & \ddots & \vdots \\ \begin{bmatrix} \int_y (B_{N_y}^y B_1^y) \int_x (B_1^x B_1^x) & \dots & \int_y (B_{N_y}^y B_1^y) \int_x (B_1^x B_{N_x}^x) \\ \vdots & \ddots & \vdots \\ \int_y (B_{N_y}^y B_1^y) \int_x (B_{N_x}^x B_1^x) & \dots & \int_y (B_{N_y}^y B_1^y) \int_x (B_{N_x}^x B_{N_x}^x) \end{bmatrix} & \dots & \begin{bmatrix} \int_y (B_{N_y}^y B_{N_y}^y) \int_x (B_1^x B_1^x) & \dots & \int_y (B_{N_y}^y B_{N_y}^y) \int_x (B_1^x B_{N_x}^x) \\ \vdots & \ddots & \vdots \\ \int_y (B_{N_y}^y B_{N_y}^y) \int_x (B_{N_x}^x B_1^x) & \dots & \int_y (B_{N_y}^y B_{N_y}^y) \int_x (B_{N_x}^x B_{N_x}^x) \end{bmatrix} \end{array} \right) \left(\begin{array}{c} \begin{bmatrix} b_{1,1} \\ \vdots \\ b_{1,N_x} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} b_{N_y,1} \\ \vdots \\ b_{N_y,N_x} \end{bmatrix} \end{array} \right) = \left(\begin{array}{c} \begin{bmatrix} \int_\Omega (B_1^y B_1^x) f \\ \vdots \\ \int_\Omega (B_1^y B_{N_x}^x) f \end{bmatrix} \\ \vdots \\ \begin{bmatrix} \int_\Omega (B_{N_y}^y B_1^x) f \\ \vdots \\ \int_\Omega (B_{N_y}^y B_{N_x}^x) f \end{bmatrix} \end{array} \right) \tag{A.3}$$

Notice that our matrix is a multiplication of two matrices, one of the block diagonal matrix:

$$
\begin{pmatrix}
\begin{bmatrix} \int_x B_1^x B_1^x & \dots & \int_x B_1^x B_{N_x}^x \\ \vdots & \ddots & \vdots \\ \int_x B_{N_x}^x B_1^x & \dots & \int_x B_{N_x}^x B_{N_x}^x \end{bmatrix} & & \\
& \ddots & \\
& & \begin{bmatrix} \int_x B_1^x B_1^x & \dots & \int_x B_1^x B_{N_x}^x \\ \vdots & \ddots & \vdots \\ \int_x B_{N_x}^x B_1^x & \dots & \int_x B_{N_x}^x B_{N_x}^x \end{bmatrix}
\end{pmatrix}
\begin{pmatrix}
\begin{bmatrix} \int_y B_1^y B_1^y & & \\ & \ddots & \\ & & \int_y B_1^y B_1^y \end{bmatrix} & \dots & \begin{bmatrix} \int_y B_1^y B_{N_y}^y & & \\ & \ddots & \\ & & \int_y B_1^y B_{N_y}^y \end{bmatrix} \\
\vdots & \ddots & \vdots \\
\begin{bmatrix} \int_y B_{N_y}^y B_1^y & & \\ & \ddots & \\ & & \int_y B_{N_y}^y B_1^y \end{bmatrix} & \dots & \begin{bmatrix} \int_y B_{N_y}^y B_{N_y}^y & & \\ & \ddots & \\ & & \int_y B_{N_y}^y B_{N_y}^y \end{bmatrix}
\end{pmatrix}
\begin{pmatrix}
\begin{bmatrix} b_{1,1} \\ \vdots \\ b_{1,N_x} \end{bmatrix} \\
\vdots \\
\begin{bmatrix} b_{N_y,1} \\ \vdots \\ b_{N_y,N_x} \end{bmatrix}
\end{pmatrix}
=
\begin{pmatrix}
\begin{bmatrix} \int_\Omega (B_1^y B_1^x) f \\ \vdots \\ \int_\Omega (B_1^y B_{N_x}^x) f \end{bmatrix} \\
\vdots \\
\begin{bmatrix} \int_\Omega (B_{N_y}^y B_1^x) f \\ \vdots \\ \int_\Omega (B_{N_y}^y B_{N_x}^x) f \end{bmatrix}
\end{pmatrix}
\tag{A.4}
$$

Also, please notice that the inverse of the block diagonal matrix is still a block diagonal matrix:

$$
\begin{pmatrix} A & & \\ & \ddots & \\ & & A \end{pmatrix}
\begin{pmatrix} A^{-1} & & \\ & \ddots & \\ & & A^{-1} \end{pmatrix}
=
\begin{pmatrix} I & & \\ & \ddots & \\ & & I \end{pmatrix}
\tag{A.5}
$$

All sub-matrices are invertible, so we end up with several identical 1D problems with multiple right-hand sides:

$$\begin{pmatrix} \begin{bmatrix} b_{1,1} \\ \vdots \\ b_{1,N_x} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} b_{N_y,1} \\ \vdots \\ b_{N_y,N_x} \end{bmatrix} \end{pmatrix} = \begin{pmatrix} \begin{bmatrix} \int\limits_x B_1^x B_1^x & \dots & \int\limits_x B_1^x B_{N_x}^x \\ \vdots & \ddots & \vdots \\ \int\limits_x B_{N_x}^x B_1^x & \dots & \int\limits_x B_{N_x}^x B_{N_x}^x \end{bmatrix}^{-1} & & \\ & \ddots & \\ & & \begin{bmatrix} \int\limits_x B_1^x B_1^x & \dots & \int\limits_x B_1^x B_{N_x}^x \\ \vdots & \ddots & \vdots \\ \int\limits_x B_{N_x}^x B_1^x & \dots & \int\limits_x B_{N_x}^x B_{N_x}^x \end{bmatrix}^{-1} \end{pmatrix} \begin{pmatrix} \begin{bmatrix} \int\limits_\Omega (B_1^y B_1^x) f \\ \vdots \\ \int\limits_\Omega (B_1^y B_{N_x}^x \end{bmatrix} \\ \dots \\ \begin{bmatrix} \int\limits_\Omega (B_{N_y}^y B_1^x) f \\ \vdots \\ \int\limits_\Omega (B_{N_y}^y B_{N_x}^x \end{bmatrix} \end{pmatrix} =$$

$$\begin{pmatrix} \begin{bmatrix} \int\limits_x B_1^x B_1^x & \dots & \int\limits_x B_1^x B_{N_x}^x \\ \vdots & \ddots & \vdots \\ \int\limits_x B_{N_x}^x B_1^x & \dots & \int\limits_x B_{N_x}^x B_{N_x}^x \end{bmatrix}^{-1} & \begin{bmatrix} \int\limits_\Omega (B_1^y B_1^x) f \\ \vdots \\ \int\limits_\Omega (B_1^y B_{N_x}^x) f \end{bmatrix} \\ \vdots & \vdots \\ \begin{bmatrix} \int\limits_x B_1^x B_1^x & \dots & \int\limits_x B_1^x B_{N_x}^x \\ \vdots & \ddots & \vdots \\ \int\limits_x B_{N_x}^x B_1^x & \dots & \int\limits_x B_{N_x}^x B_{N_x}^x \end{bmatrix}^{-1} & \begin{bmatrix} \int\limits_\Omega (B_{N_y}^y B_1^x) f \\ \vdots \\ \int\limits_\Omega (B_{N_y}^y B_{N_x}^x) f \end{bmatrix} \end{pmatrix} = \begin{pmatrix} \begin{bmatrix} t_{1,1} \\ \vdots \\ t_{1,N_x} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} t_{N_y,1} \\ \vdots \\ t_{N_y,N_x} \end{bmatrix} \end{pmatrix} \tag{A.6}$$

Then, we complete the following re-ordering in the block system

$$\left(\begin{bmatrix} b_{1,1} \\ \vdots \\ b_{1,N_x} \end{bmatrix} \cdots \begin{bmatrix} b_{N_y,1} \\ \vdots \\ b_{N_y,N_x} \end{bmatrix}\right) \rightarrow \left(\begin{bmatrix} b_{1,1} \\ \vdots \\ b_{N_y,1} \end{bmatrix} \cdots \begin{bmatrix} b_{1,N_x} \\ \vdots \\ b_{N_y,N_x} \end{bmatrix}\right) \left(\begin{bmatrix} t_{1,1} \\ \vdots \\ t_{1,N_x} \end{bmatrix} \cdots \begin{bmatrix} t_{N_y,1} \\ \vdots \\ t_{N_y,N_x} \end{bmatrix}\right) \rightarrow \left(\begin{bmatrix} t_{1,1} \\ \vdots \\ t_{N_y,1} \end{bmatrix} \cdots \begin{bmatrix} t_{1,N_x} \\ \vdots \\ t_{N_y,N_x} \end{bmatrix}\right) \tag{A.7}$$

to get another block diagonal problem:

$$\begin{pmatrix} \begin{bmatrix} \int\limits_y B_1^y B_1^y & \cdots & \int\limits_y B_1^y B_{N_y}^y \\ \vdots & \ddots & \vdots \\ \int\limits_y B_{N_y}^y B_1^y & \cdots & \int\limits_y B_{N_y}^y B_{N_y}^y \end{bmatrix} & & \\ & \ddots & \\ & & \begin{bmatrix} \int\limits_y B_1^y B_1^y & \cdots & \int\limits_y B_1^y B_{N_y}^y \\ \vdots & \ddots & \vdots \\ \int\limits_y B_{N_y}^y B_1^y & \cdots & \int\limits_y B_{N_y}^y B_{N_y}^y \end{bmatrix} \end{pmatrix} \begin{pmatrix} \begin{bmatrix} b_{1,1} \\ \vdots \\ b_{N_y,1} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} b_{1,N_x} \\ \vdots \\ b_{N_y,N_x} \end{bmatrix} \end{pmatrix} = \begin{pmatrix} \begin{bmatrix} t_{1,1} \\ \vdots \\ t_{N_y,1} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} t_{1,N_x} \\ \vdots \\ t_{N_y,N_x} \end{bmatrix} \end{pmatrix} \tag{A.8}$$

which implies again several identical 1D problems with different right-hand-sides.

$$\begin{pmatrix} \begin{bmatrix} b_{1,1} \\ \vdots \\ b_{N_y,1} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} b_{1,N_x} \\ \vdots \\ b_{N_y,N_x} \end{bmatrix} \end{pmatrix} = \begin{pmatrix} \begin{bmatrix} \int\limits_y B_1^y B_1^y & \cdots & \int\limits_y B_1^y B_{N_y}^y \\ \vdots & \ddots & \vdots \\ \int\limits_y B_{N_y}^y B_1^y & \cdots & \int\limits_y B_{N_y}^y B_{N_y}^y \end{bmatrix}^{-1} \begin{bmatrix} t_{1,1} \\ \vdots \\ t_{N_y,1} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} \int\limits_y B_1^y B_1^y & \cdots & \int\limits_y B_1^y B_{N_y}^y \\ \vdots & \ddots & \vdots \\ \int\limits_y B_{N_y}^y B_1^y & \cdots & \int\limits_y B_{N_y}^y B_{N_y}^y \end{bmatrix}^{-1} \begin{bmatrix} t_{1,N_x} \\ \vdots \\ t_{N_y,N_x} \end{bmatrix} \end{pmatrix} \tag{A.9}$$

To complete our algebra, we denote

$$\begin{pmatrix} \begin{bmatrix} d_{1,1} \\ \vdots \\ d_{N_y,1} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} d_{1,N_x} \\ \vdots \\ d_{N_y,N_x} \end{bmatrix} \end{pmatrix} = \begin{pmatrix} \begin{bmatrix} \int_y B_1^y B_1^y & \dots & \int_y B_1^y B_{N_y}^y \\ \vdots & \ddots & \vdots \\ \int_y B_{N_y}^y B_1^y & \dots & \int_y B_{N_y}^y B_{N_y}^y \end{bmatrix}^{-1} & \begin{bmatrix} t_{1,1} \\ \vdots \\ t_{N_y,1} \end{bmatrix} \\ & \vdots & \vdots \\ \begin{bmatrix} \int_y B_1^y B_1^y & \dots & \int_y B_1^y B_{N_y}^y \\ \vdots & \ddots & \vdots \\ \int_y B_{N_y}^y B_1^y & \dots & \int_y B_{N_y}^y B_{N_y}^y \end{bmatrix}^{-1} & \begin{bmatrix} t_{1,N_x} \\ \vdots \\ t_{N_y,N_x} \end{bmatrix} \end{pmatrix} \tag{A.10}$$

and reorder $\begin{pmatrix} \begin{bmatrix} b_{1,1} \\ \vdots \\ b_{N_y,1} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} b_{1,N_x} \\ \vdots \\ b_{N_y,N_x} \end{bmatrix} \end{pmatrix}$ back to $\begin{pmatrix} \begin{bmatrix} b_{1,1} \\ \vdots \\ b_{1,N_x} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} b_{N_y,1} \\ \vdots \\ b_{N_y,N_x} \end{bmatrix} \end{pmatrix}$ to get finally $\begin{pmatrix} \begin{bmatrix} b_{1,1} \\ \vdots \\ b_{1,N_x} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} b_{N_y,1} \\ \vdots \\ b_{N_y,N_x} \end{bmatrix} \end{pmatrix} = \begin{pmatrix} \begin{bmatrix} d_{1,1} \\ \vdots \\ d_{N_y,1} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} d_{1,N_x} \\ \vdots \\ d_{N_y,N_x} \end{bmatrix} \end{pmatrix}$

This algorithm for two-dimensional L^2 projection with N unknowns involves solutions of two one-dimensional systems with $N^{\frac{1}{2}}$ unknowns and $N^{\frac{1}{2}}$ right hand sides. This algorithm for three-dimensional L^2 projection with N unknowns involves solutions of two one-dimensional systems with $N^{\frac{1}{3}}$ unknowns and $N^{\frac{1}{3}}$ right hand sides.

Appendix B

Complexity analysis of the sequential isogeometric L^2 projection algorithm

B.0.1. Integration over one element

Every element is approximated by a set of polynomials in each direction where p is the order, and there are $p+1$ B-splines over the element. We denote p_x as the degree in x direction and p_y and p_z as degrees in other directions. The integration of the right hand side requires using Gaussian quadrature with $(p_x+1)(p_y+1)(p_z+1)$ points. The integral over each element is:

$$\sum_{m=1}^{(p_x+1)(p_y+1)(p_z+1)} w_m B_x^i(x_m) B_y^j(y_m) B_z^k(z_m) f(x_m, y_m, z_m) dxdydz \tag{B.1}$$

where w_m denotes the Gaussian quadrature weights, B_x^i, B_y^j, B_z^k denotes the B-spline basis functions in x, y, and z directions, respectively, computed at x_m, y_m, z_m Gaussian quadrature points, and we have $i = 1, \ldots, p_x + 1$, $j = 1, \ldots, p_y + 1$ and $k = 1, \ldots, p_z + 1$ entries to compute. Assume that for $d = 1, \ldots, (p_x+1)(p_y+1)(p_z+1)$ counting value at given point for given element and function f costs $\Phi^f((p_x+1)^2(p_y+1)^2(p_z+1)^2)$ arithmetic operations where Φ^f is the function depending on f.
The formula for Φ^f depends on the form of f. If f is given by a prescribed formula, then cost of computing a value of f is constant and Φ^f is constant. Otherwise when f is given by a combination of $B-splines$

$$f = \sum_{o=1}^{p_x+1} \sum_{q=1}^{p_y+1} \sum_{r=1}^{p_z+1} B_x^o B_y^q B_z^r f_{oqr} \tag{B.2}$$

then

$$\Phi^f(x_m) = (p_x+1)(p_y+1)(p_z+1) \tag{B.3}$$

and total cost will be

$$(p_x+1)^3(p_y+1)^3(p_z+1)^3 \tag{B.4}$$

In the following part of the paper we assume that f is prescribed by a given formula, and so the cost of computation a value of f at given point is constant.

B.0.2. Integration over all elements

Since we have a mesh of $N_x \times N_y \times N_z$ elements (where N_x, N_y, N_z denotes the number of elements in the x, y and z direction, respectively) the total cost of integration will be

$$(p_x+1)^2(p_y+1)^2(p_z+1)^2 N_x N_y N_z \Phi^f \tag{B.5}$$

with computational complexity of

$$\mathcal{O}(p_x^2 p_y^2 p_z^2 N_x N_y N_z) \tag{B.6}$$

B.0.3. Solution

In each step of the algorithm we LU factorize a banded matrix resulting from one dimensional B-spline basis function of order p. Let N be the number of elements in one direction. Then, the banded matrix M^N of size N with $2p+1$ diagonal blocks can be LU factorized in $\mathcal{O}(p^2 N)$ steps.
When solving problem in the x direction we have to LU factorize matrix M^{N_x} of size N_x with $2p_x+1$ diagonal blocks and we have $N_y \times N_z$ right hand sides, each one of size N_x.

$$\mathcal{O}\left(N_x p_x^2 N_y N_z\right) \tag{B.7}$$

The solution complexity over y and z directions can be estimated in analogous way as

$$\mathcal{O}\left(N_y p_y^2 N_x N_z\right) \tag{B.8}$$

and

$$\mathcal{O}\left(N_z p_z^2 N_x N_y\right) \tag{B.9}$$

this results in computational complexity

$$\mathcal{O}\left((p_x^2+p_y^2+p_z^2)(N_x N_y N_z)\right) \tag{B.10}$$

B.0.4. Reorder data

After processing data in the x-direction we need to perform the reorder of data processing along the y direction. Similar reordering applies after processing data in the y-direction and before processing data in the z-direction. The computational complexity of each of the two reorders is equal to

$$\mathcal{O}\left(N_x N_y N_z p_x p_y p_z\right) \tag{B.11}$$

B.0.5. Total complexity

From the discussion above, we conclude that we can construct isogeometric projection solver with the total cost

$$\left(p_x^2 p_y^2 p_z^2 N_x N_y N_z\right) t_{comp} + \left(p_x^2+p_y^2+p_z^2\right)\left(N_x N_y N_z\right) t_{comp} + + \left(p_x p_y p_z N_x N_y N_z\right) t_{comp} \tag{B.12}$$

for arbitrary polynomial orders p_x, p_y, p_z, dimension sizes N_x, N_y, N_z where t_{comp} is the cost of processing a single FLOAT.

Assuming

$$N_x = N_y = N_z = N^{1/3},\ p_x = p_y = p_z = p \tag{B.13}$$

we have the following cost

$$\left(p^6 N + p^2 N + p^3 N\right) t_{comp} \tag{B.14}$$

which implies the computational complexity

$$\mathcal{O}\left(p^6 N\right) \tag{B.15}$$

Appendix C

Derivation of element matrices and right-hand-side vector for the isogeometric L^2 projection problem

This section presents a weak form of the classical L^2 projection problem, used as a model problem to test the efficiency of our parallel integration algorithm. The L^2-projection problem in the weak form is the following: We are looking for the $u \in V \subset L^2$, being the orthogonal L^2 projection of picewise constant function F from $L^2(\Omega)$ on the space of B-splines V, which results in the condition $(F-u,v)=0, \forall v \in V$, where $(L^2(\Omega))^2 \ni (u,v) \to \int_\Omega u\,v\mathrm{d}x \in \mathbb{R}$ stands for the scalar product in $L^2(\Omega)$. Observing that $V-u=V$ we can obtain the L^2-projection problem in the more convenient form:

$$(u,v)=(F,v)\ \forall v \in V, \quad V=span\{B_{i,j;p}\}_{i=1,\ldots,N_x,j=1,\ldots,N_y} \tag{C.1}$$

where $B_{i,j;p}$ are B-spline basis functions of a given order p [29]. The B-spline order p is uniform, and constant for all basis functions.

If we select a basis $\{B_{i,j;p}\}_{i=1,\ldots,N_x,j=1,\ldots,N_y}$ in V, then an arbitrary $v \in V$ can be represented by $v=\sum_{k=1,\ldots,N_x,l=1,\ldots,N_y} B_{k,l;p}b_{k,l}$ and the L^2-projection problem can be rewritten in the form

$$\sum_{k=1,\ldots,N_x,l=1,\ldots,N_y} b_{k,l}(u-F,B_{k,l;p})=0\ \forall\{b_{k,l}\} \tag{C.2}$$

Because $\{b_{k,l}\}$ is arbitrary, then (C.2) is satisfied if and only if $R^{N_xN_y} \ni (u-F,B_{k,l;p})=0$ as the unique vector in $R^{N_xN_y}$ orthogonal to all others. Then, we obtain a system of equations

$$(u,B_{k,l;p})=(F,B_{k,l;p})\ k=1,\ldots,N_x, l=1,\ldots,N_y \tag{C.3}$$

moreover, finally using the representation of

$$u=\sum_{i,j} B_{i,j;p}a_{i,j} \tag{C.4}$$

we obtain

$$\sum_{i=1,\ldots,N_x,j=1,\ldots,N_y} a_{i,j}(B_{i,j;p},B_{k,l;p})=(F,B_{k,l;p})\ k=1,\ldots,N_x, l=1,\ldots,N_y \tag{C.5}$$

List of figures

List of tables

Abbreviations

1D One-dimensional

2D Two-Dimensional

3D Three-Dimensional

ADS Alternating Directions Solver

CAD Computer Aided Design

CAE Computer Aided Engineering

CPU Central Processing Unit

FEM Finite Element Method

GPGPU General-Purpose computing on Graphics Processing Units

GPU Graphics Processing Unit

IGA Isogeometric Analysis

MPI Message Passing Interface

MUMPS MUltifrontal Massively Parallel sparse direct Solver

PaStiX Parallel Sparse matriX package

PCAM Partitioning Communication Agglomeration Mapping

PETIGA A framework for high performance Isogeometric Analysis

PETSc Portable, Extensible Toolkit for Scientific Computation

TACC Texas Advanced Computing Center

Bibliography

[1] http://www.mcs.anl.gov/petsc.

[2] H. AbouEisha, M. Moshkov, V. Calo, M. Paszyński, D. Goik, and K. Jopek. Dynamic programming algorithm for generation of optimal elimination trees for multi-frontal direct solver over h-refined grids. *Procedia Computer Science*, 29:947–959, 2014.

[3] I. Akkerman, Y. Bazilevs, V. M. Calo, T. J. R. Hughes, and S. Hulshoff. The role of continuity in residual-based variational multiscale modeling of turbulence. *Computational Mechanics*, 41:371–378, 2008.

[4] M. Alotaibi, V. M. Calo, Y. Efendiev, J. Galvis, and M. Ghommem. Global-local nonlinear model reduction for flows in heterogeneous porous media. *Computer Methods in Applied Mechanics and Engineering*, 292:122–137, 2015.

[5] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent. A fully synchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23:15–41, 2001.

[6] P. R. Amestoy, I. S. Duff, and J.-Y. L'Excellent. Multifrontal parallel distributed symmetric and unsymmetric solvers. *Computer Methods in Applied Mechanics and Engineering*, 184:501–520, 2000.

[7] P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet. Hybrid scheduling for the parallel solution of linear systems. *Parallel Computing*, 32:136–156, 2006.

[8] D. N. Arnold, R. S. Falk, and R. Winther. Multigrid in h(div) and h(curl). *Numerische Mathematik*, 85:197–217, 2000.

[9] S. Balay, S. Abhyankar, M. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, V. Eijkhout, W. Gropp, D. Karpeyev, D. Kaushik, M. Knepley, L. Curfman McInnes, K. Rupp, B. Smith, S. Zampini, and H. Zhang. *PETSc Users Manual*. Argonne National Laboratory, http://www.mcs.anl.gov/petsc/petsc-current/docs/manual.pdf, 2013. Revision 3.7.

[10] 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, and H. Zhang. *PETSc Web Page*. http://www.mcs.anl.gov/petsc, 2014.

[11] S. Balay, W. D. Gropp, L. Curfman McInnes, and B. F. Smith. Efficient management of parallelism in object-oriented numerical software libraries. *Modern Software Tools in Scientific Computing*, pages 163–202, 1997.

[12] Y. Bazilevs, L. Beirao da Veiga, J. A. Cottrell, T. J. R. Hughes, and G. Sangalli. Isogeometric analysis: Approximation, stability and error estimates for h-refined meshes. *Mathematical Models and Methods in Applied Sciences*, 16:1031–1090, 2006.

[13] Y. Bazilevs, V. M. Calo, J. A. Cottrell, T. J. R. Hughes, A. Reali, and G. Scovazzi. Variational multiscale residual-based turbulence modeling for large eddy simulation of incompressible flows. *Computer Methods in Applied Mechanics and Engineering*, 197:173–201, 2007.

[14] Y. Bazilevs, V. M. Calo, Y. Zhang, and T. J. R. Hughes. Isogeometric fluid-structure interaction analysis with applications to arterial blood flow. *Computational Mechanics*, 38:310–322, 2006.

[15] Y. Bazilevs, C. Michler, V. M. Calo, and T. J. R. Hughes. Isogeometric variational multiscale modeling of wall-bounded turbulent flows with weakly enforced boundary conditions on unstretched meshes. *Computer Methods in Applied Mechanics and Engineering*, 199:780–790, 2010.

[16] D. J. Benson, Y. Bazilevs, E. De Luycker, M.-C. Hsu, M. Scott, T. J. R. Hughes, and T. Belytschko. A generalized finite element formulation for arbitrary basis functions: From isogeometric analysis to xfem. *International Journal for Numerical Methods in Engineering*, 83:765–785, 2010.

[17] D. J. Benson, Y. Bazilevs, M.-C. Hsu, and T. J. R. Hughes. A large deformation, rotation-free, isogeometric shell. *Computer Methods in Applied Mechanics and Engineering*, 200:1367–1378, 2011.

[18] L. M. Bernal, V. M. Calo, N. Collier, G. A. Espinosa, F. Fuentes, and J. C. Mahecha. Isogeometric analysis of hyperelastic materials using petiga. *Procedia Computer Science*, 18:1604–1613, 2013.

[19] V. Bientinesi, P. amd Eijkhout, K. Kim, J. Kurtz, and van de Geijn R. Sparse direct factorizations through unassembled hyper-matrices. *Computer Methods in Applied Mechanics and Engineering*, 199:430–438, 2010.

[20] G. Birkhoff, R. Varga, and Y. D. Alternating direction implicit methods. *Advances in Computers*, 3:189–273, 1962.

[21] 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, and R. C. Whaley. *ScaLAPACK Users' Guide*. Society for Industrial and Applied Mathematics, 1997.

[22] A. Buffa, H. Harbrecht, A. Kunoth, and G. Sangalli. Bpx-preconditioning for isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 265:63–70, 2013.

[23] V. M. Calo, N. F. Brasher, Y. Bazilevs, and T. J. R. Hughes. Multiphysics model for blood flow and drug transport with application to patient-specific coronary artery flow. *Computational Mechanics*, 43:161–177, 2008.

[24] V. M. Calo, N. O. Collier, D. Pardo, and M. R. Paszyński. Computational complexity and memory usage for multi-frontal direct solvers used in p finite element analysis. *Procedia Computer Science*, 4:1854–1861, 2011.

[25] V. M. Calo, H. Gomez, Y. Bazilevs, G. Johnson, and T. J. R. Hughes. Simulation of engineering applications using isogeometric analysis. *Proceedings of Tera Grid*, 2008.

[26] K. Chang, T. J. R. Hughes, and V. M. Calo. Isogeometric variational multiscale large-eddy simulation of fully-developed turbulent flow over a wavy wall. *Computers & Fluids*, 68:94–104, 2012.

[27] N. Collier, L. Dalcin, D. Pardo, and V. M. Calo. The cost of continuity: Performance of iterative solvers on isogeometric finite elements. *SIAM Journal on Scientific Computing*, 35:A767–A784, 2013.

[28] N. Collier, D. Pardo, L. D. Dalcin, M. Paszyński, and V. M. Calo. The cost of continuity: A study of the performance of isogeometric finite elements using direct solvers. *Computer Methods in Applied Mechanics and Engineering*, 213-216:353–361, 2012.

[29] J. A. Cottrel, T. J. R. Hughes, and Y. Bazilevs. *Isogeometric Analysis: Toward Integration of CAD and FEA*. Wiley, 2009.

[30] J. A. Cottrell, T. R. Hughes, and Y. Bazilevs. *Isogeometric Analysis: Toward Integration of CAD and FEA*. John Wiley & Sons, 2009.

[31] L. Dedè, M. J. Borden, and T. J. R. Hughes. Isogeometric analysis for topology optimization with a phase field model. *Archives of Computational Methods in Engineering*, 19:427–465, 2012.

[32] L. Dedè, T. J. R. Hughes, S. Lipton, and V. M. Calo. Structural topology optimization with isogeometric analysis in a phase field approach. *16th US National Conference on Theoretical and Applied Mechanics*, 2010.

[33] L. Demkowicz. *Computing with hp-ADAPTIVE FINITE ELEMENTS: Volume 1 One and Two Dimensional Elliptic and Maxwell Problems*. Chapman & Hall/CRC Applied Mathematics & Nonlinear Science. Chapmann & Hall / CRC, 2006.

[34] L. Demkowicz, J. Kurtz, D. Pardo, M. Paszyński, W. Rachowicz, and A. Zdunek. *Computing with hp-ADAPTIVE FINITE ELEMENTS: Volume 2 Frontiers: Three Dimensional Elliptic and Maxwell Problems with Applications*. Chapman & Hall/CRC Applied Mathematics & Nonlinear Science. Chapmann & Hall / CRC, 2007.

[35] V. Diekert and G. Rozenberg. *The Book of Traces*. World Scientific, 1995.

[36] M. Dőrfel, B. Júttler, and B. Simeon. Adaptive isogeometric analysis by local h-refinement with t-splines. *Computer Methods in Applied Mechanics and Engineering*, 199:264–275, 2010. Computational Geometry and Analysis.

[37] J. Douglas and H. H. Rachford. On the numerical solution of heat conduction problems in two and three space variables. *Transactions of American Mathematical Society*, 82:421–439, 1956.

[38] R. Duddu, L. L. Lavier, T. J. R. Hughes, and V. M. Calo. A finite strain eulerian formulation for compressible and nearly incompressible hyperelasticity using high-order b-spline finite elements. *International Journal for Numerical Methods in Engineering*, 89:762–785, 2012.

[39] I. S. Duff and J. K. Reid. The multifrontal solution of indefinite sparse symmetric linear. *ACM Transactions on Mathematical Software*, 9:302–325, 1983.

[40] I. S. Duff and J. K. Reid. The multifrontal solution of unsymmetric sets of linear equations. *SIAM Journal on Scientific and Statistical Computing*, 5:633–641, 1984.

[41] A. El maliki, M. Fortin, N. Tardieu, and A. Fortin. Iterative solvers for 3d linear and nonlinear elasticity problems: Displacement and mixed formulations. *International Journal for Numerical Methods in Engineering*, 83:1780–1802, 2010.

[42] I. Foster. *Designing and building parallel programs*. 1995.

[43] L. Gao and V. Calo. Fast isogeometric solvers for explicit dynamics. *Computer Methods in Applied Mechanics and Engineering*, 274:19–41, 2014.

[44] L. Gao and V. Calo. Preconditioners based on the alternating-direction-implicit algorithm for the 2d steady-state diffusion equation with orthotropic heterogenous coefficients. *Journal of Computational and Applied Mathematics*, 273:274–295, 2015.

[45] L. Gao and V. M. Calo. Fast isogeometric solvers for explicit dynamics. *Computer Methods in Applied Mechanics and Engineering*, 274:19–41, 2014.

[46] L. Gao and V. M. Calo. Preconditioners based on the alternating-direction-implicit algorithm for the 2d steady-state diffusion equation with orthotropic heterogeneous coefficients. *Journal of Computational and Applied Mathematics*, 273:274–295, 2015.

[47] P. Geng, J. T. Oden, and R. A. van de Geijn. A parallel multifrontal algorithm and its implementation. *Computer Methods in Applied Mechanics and Engineering*, 149:289–301, 1997.

[48] D. Goik, K. Jopek, M. Paszyński, A. Lenharth, D. Nguyen, and K. Pingali. Graph grammar based multi-thread multi-frontal direct solver with galois scheduler. *Procedia Computer Science*, 29:960–969, 2014.

[49] H. Gomez, V. M. Calo, Y. Bazilevs, and T. J. R. Hughes. Isogeometric analysis of the cahn-hilliard phase-field model. *Computer Methods in Applied Mechanics and Engineering*, 197:4333–4352, 2008.

[50] H. Gomez, T. J. R. Hughes, X. Nogueira, and V. M. Calo. Isogeometric analysis of the isothermal navier-stokes-korteweg equations. *Computer Methods in Applied Mechanics and Engineering*, 199:1828–1840, 2010.

[51] P. Hénon, P. Ramet, and J. Roman. Pastix: A high-performance parallel direct solver for sparse symmetric definite systems. *Parallel Computing*, 28:301–321, 2002.

[52] R. Hiptmair. Multigrid method for maxwell's equations. *SIAM Journal of Numerical Analysis*, 36:204–225, 1998.

[53] S. S. Hossain, S. F. A. Hossainy, Y. Bazilevs, V. M. Calo, and T. J. R. Hughes. Mathematical modeling of coupled drug and drug-encapsulated nanoparticle transport in patient-specific coronary artery walls. *Computational Mechanics*, 49:213–242, 2012.

[54] M.-C. Hsu, I. Akkerman, and Y. Bazilevs. High-performance computing of wind turbine aerodynamics using isogeometric analysis. *Computers & Fluids*, 49:93–100, 2011.

[55] O. Iliev, R. Lazarov, and J. Willems. Variational multiscale finite element method for flows in highly porous media. *Multiscale Model. Simul.*, 9:1350–1372, 2011.

[56] B. M. Irons. A frontal solution program for finite-element analysis. *International Journal for Numerical Methods in Engineering*, 2:5–32, 1970.

[57] K. Kuźnik, M. Paszyński, and V. Calo. Grammar based multi-frontal solver for isogeometric analysis in 1d. *Computer Science*, 14:589–613, 2013.

[58] X. Li, J. Demmel, J. Gilbert, i. Grigori, M. Shao, and I. Yamazaki. *SuperLU Users' Guide*. Lawrence Berkeley National Laboratory, http://crd.lbl.gov/ xiaoye/SuperLU/, 1999.

[59] X. S. Li. An overview of superlu: Algorithms, implementation, and user interface. *TOMS Transactions on Mathematical Software*, 31:302–325, 2005.

[60] L. Lin, C. Yang, J. Lu, L. Ying, and E. Weinan. A fast parallel algorithm for selected inversion of structured sparse matrices with application to 2d electronic structure calculations. *SIAM Journal on Scientific Computing*, 33:1329–1351, 2011.

[61] P. Obrok, P. Pierzchała, A. Szymczak, and M. Paszyński. Graph grammar-based multi-thread multi-frontal parallel solver with trace theory-based scheduler. *Procedia Computer Science*, 1:1993–2001, 2010.

[62] A. Paszyńska, , E. Grabska, and M. Paszyński. A graph grammar model of the hp adaptive three dimensional finite element method. part i. *Fundamenta Informaticae*, 114:149–182, 2012.

[63] A. Paszyńska, , E. Grabska, and M. Paszyński. A graph grammar model of the hp adaptive three dimensional finite element method. part ii. *Fundamenta Informaticae*, 114:183–201, 2012.

[64] A. Paszyńska, M. Paszyński, K. Jopek, M. Woźniak, D. Goik, P. Gurgul, H. AbouEisha, M. Moshkov, V. M. Calo, A. Lenharth, D. Nguyen, and K. Pingali. Quasi-optimal elimination trees for 2d grids with singularities. *Scientific Programming*, 2015, 2015.

[65] M. Paszyński. On the parallelization of self-adaptive hp-finite element methods part ii. partitioning communication agglomeration mapping (pcam) analysis. *Fundamenta Informaticae*, 93:435–457, 2009.

[66] M. Paszyński. Minimizing the memory usage with parallel out-of-core multi-frontal direct solver. *Computer Assisted Methods in Engineering and Science*, 20:15–41, 2013.

[67] M. Paszyński, T. Jurczyk, and D. Pardo. Multi-frontal solver for simulations of linear elasticiy coupled with acoustics. *Computer Science*, 12:85–102, 2011.

[68] M. Paszyński, D. Pardo, and A. Paszyńska. Parallel multi-frontal solver for p adaptive finite element modeling of multi-physics computational problems. *Journal of Computational Science*, 1:48–54, 2010.

[69] M. Paszyński, D. Pardo, C. Torres-Verdín, L. Demkowicz, and C. V. M. A parallel direct solver for self-adaptive hp-finite element method. *Journal of Parallel and Distributed Computing*, 70:270–281, 2010.

[70] M. Paszyński and R. Schaefer. Graph grammar-driven parallel partial differential equation solver. *Concurrency and Computation Practice and Experience*, 22:1063–1097, 2010.

[71] D. W. Peaceman and J. H. H. Rachford, H. H. The numerical solution of parabolic and elliptic differential equation. *Journal of the Society for Industrial and Applied Mathematics*, 3:28–41, 1955.

[72] P. G. Schmitz and L. Ying. A fast direct solver for elliptic problems on general meshes in 2d. *Journal of Computational Physics*, pages 1314–1338, 2012.

[73] P. G. Schmitz and L. Ying. A fast multifrontal solver for 3d elliptic problems using hierarchical matrices. *Journal of Computational Physics*, 2013. submitted.

[74] B. Strug, A. Paszyńska, M. Paszyński, and G. E. Using a graph grammar system in the finite element method. *International Journal of Applied Mathematics and Computer Science*, 23:839–853, 2013.

[75] A. Szymczak, M. Paszyński, D. Pardo, and A. Paszyńska. Petri nets modeling of dead-end refinement problems in a 3d anisotropic hp-adaptive finite element method. *Computing and Informatics*, 34:425–457, 2015.

[76] Texas Advanced Computing Center, https://portal.tacc.utexas.edu/user-guides/stampede. *STAMPEDE User Guide*, 2016.

[77] E. L. Wachspress and G. J. Habetler. An alternating-direction-implicit iteration technique. *Journal of Society of Industrial and Applied Mathematics*, 8:403–423, 1960.

[78] M. Woźniak. Task dependency graph based scheduler for fast gpu integration for isogeometric finite element method solvers. *Journal of Computational Science*, 11:145–152, 2015.

[79] M. Woźniak, K. Kuźnik, M. Paszyński, V. M. Calo, and D. Pardo. Computational cost estimates for parallel shared memory isogeometric multi-frontal solvers. *Computers & Mathematics with Applications*, 67:1864–1883, 2014.

[80] M. Woźniak, M. Łoś, M. Paszyński, L. Dalcin, and V. M. Calo. Parallel three dimensional isogeometric l^2 projection solver. accepted to CAI.

[81] M. Woźniak, M. Paszyński, D. Pardo, and L. C. V. M. Dalcin. Computational cost of isogeometric multi-frontal solvers on parallel distributed memory machines. *Computer Methods in Applied Mechanics and Engineering*, 284:971–987, 2015.

Maciej Woźniak, mgr inż.
WIEiT-ki
Wydział Informatyki, Elektroniki i Telekomunikacji
Katedra Informatyki

1. *Alternating directions solver for isogeometric simulations of non-linear problems* / Marcin Łoś, Maciej WOŹNIAK, Maciej PASZYŃSKI // W: *GEASC Global Engineering & Applied Science Conference ; SEDT International Symposium on Electrical, Electronic Engineering and Digital Technology ; ILSBE International Conference on Life Science and Biological Engineering [Dokument elektroniczny]* : the joint conference : December, 2015 Tokyo, Japan : conference proceedings. — Wersja do Windows. — Dane tekstowe. — [Tokyo : s. n.], [2015]. — 1 dysk Flash. — dod. ISBN: 978-986-5654-05-4 (SEDT), ISBN: 978-986-5654-0407 (ILSBE). — e-ISBN: 978-986-5654-31-3 (G. — S. 296–305. — Wymagania systemowe: Adobe Reader. — Bibliogr. s. 305, Abstr.. — publikacja zamieszczona w GEASC

brak Impact Factor — brak punktacji MNiSW

2. *Alternating directions solver for isogeometric simulations of non-linear problems* / Marcin ŁOŚ, Maciej WOŹNIAK, Maciej PASZYŃSKI // *International Journal of Computer Science and Engineering* ; ISSN 2278-9960. — 2016 vol. 5 iss. 2, s. 99-107. — Bibliogr. s. 107, Abstr.

brak Impact Factor — brak punktacji MNiSW

3. *Application of projection-based interpolation algorithm for non-stationary problem* / Maciej WOŹNIAK, Maciej PASZYŃSKI // *Computer Science* ; ISSN 1508-2806. — 2016 vol. 17 no. 3, s. 297–319. — Bibliogr. s. 318–319, Abstr.

brak Impact Factor — punktacja (lista B czasopism MNiSW, 2016): **12.000**

4. *Comparison of the structure of equation systems and the GPU multifrontal solver for finite difference, collocation and finite element method* / P. Lipski, M. WOŹNIAK, M. PASZYŃSKI // *Procedia Computer Science [Dokument elektroniczny]. - Czasopismo elektroniczne* ; ISSN 1877-0509. — 2015 vol. 51, s. 1072–1081. — Bibliogr. s. 1081, Abstr.. — ICCS 2015 : International Conference On Computational Science : Computational Science at the Gates of Nature : June 1–3, 2015 in Reykjavík, Iceland. — tekst: `http://www.sciencedirect.com/science/article/pii/S1877050915010741/pdf?md5=b4a6f96d61999ce2f4845ed2fb4a18d1&pid=1-s2.0-S1877050915010741-main.pdf`

brak Impact Factor — punktacja (lista czasopism MNiSW, 2015): **15.000**

5. *Computational cost estimates for parallel shared memory isogeometric multi-frontal solvers* / M. WOŹNIAK, K. Kuźnik, M. PASZYŃSKI, V. M. Calo, D. Pardo // *Computers and Mathematics with Applications* ; ISSN 0898-1221. — 2014 vol. 67 iss. 10, s. 1864–1883. — Bibliogr. s. 1882–1883, Abstr.. — tekst: `http://vls2.icm.edu.pl/cgi-bin/sciserv.pl?collection=elsevier&journal=08981221&issue=v67i0010&article=1864_ccefpsmims&form=pdf&file=file.pdf`

1.697 — punktacja (lista A czasopism MNiSW, 2014): **40.000**

6. *Computational cost of isogeometric multi-frontal solvers on parallel distributed memory machines* / Maciej WOŹNIAK, Maciej PASZYŃSKI, David Pardo, Lisandro Dalcin, Victor Manuel Calo // *Computer Methods in Applied Mechanics and Engineering* ; ISSN 0045-7825. — 2015 vol. 284, spec. iss.: Isogeometric analysis, s. 971–987. — Bibliogr. s. 986–987, Abstr.. — tekst: `http://www.sciencedirect.com/science/article/pii/S0045782514004460/pdfft?md5=d338712f438c0ca101e7aae45e4bbe5c&pid=1-s2.0-S0045782514004460-main.pdf`

3.467 — punktacja (lista A czasopism MNiSW, 2015): **45.000**

7. *Dynamics with matrices possessing Kronecker product structure* / M. ŁOŚ, M. WOŹNIAK, M. PASZYŃSKI, L. Dalcin, V. M. Calo // *Procedia Computer Science [Dokument elektroniczny]. - Czasopismo elektroniczne* ; ISSN 1877-0509. — 2015 vol. 51, s. 286–295. — Bibliogr. s. 294, Abstr.. — ICCS 2015 : International Conference on Computational Science : Computational Science at the Gates of Nature : 1–3 June 2015, Reykjavík, Iceland. — tekst: `http://www.sciencedirect.com/science/article/pii/S1877050915010510/pdf?md5=2fb6fa013dff7f43ea61f0461820aac7&pid=1-s2.0-S1877050915010510-main.pdf`

brak Impact Factor — punktacja (lista czasopism MNiSW, 2015): **15.000**

8. *Explicit method solver based on alternating direction isogeometric L2 projections* / Maciej R. PASZYŃSKI, Maciej WOŹNIAK, Lisandro D. Dalcin, Victor M. Calo // W: *WCCM XI ; ECCM V ; ECFD VI [Dokument elektroniczny]* : 11th World Congress on Computational Mechanics ; 5th European Conference on Computational Mechanics : 6th European Conference on Computational Fluid Dynamics : Barcelona, Spain, 20–25 July 2014. — Wersja do Windows. — Dane tekstowe. — [Barcelona : International Center of Numerical Methods Engineering], 2014. — e-ISBN: 978-84-942844-7-2. — S. 1–2. — Bibliogr. s. 2. — tekst: `http://www.wccm-eccm-ecfd2014.org/admin/files/fileabstract/a499.pdf`

brak Impact Factor — brak punktacji MNiSW

9. *Fast GPU integration algorithm for isogeometric finite element method solvers using task dependency graphs* / Maciej WOŹNIAK // *Journal of Computational Science* ; ISSN 1877-7503. — 2015 vol. 11, s. 145–152. — Bibliogr. s. 151–152, Abstr.. — tekst: `http://www.sciencedirect.com.atoz.wbg2.bg.agh.edu.pl/science/article/pii/S1877750315000253/pdfft?md5=5e80a98561b590825c8851bea23fa5ca&pid=1-s2.0-S1877750315000253-main.pdf`

1.078 — punktacja (lista A czasopism MNiSW, 2015): **30.000**

10. *Fast parallel integration for three dimensional Discontinuous Petrov Galerkin method* / Maciej WOŹNIAK, Marcin ŁOŚ, Maciej PASZYŃSKI, Leszek Demkowicz // *Procedia Computer Science [Dokument elektroniczny]. - Czasopismo elektroniczne* ; ISSN 1877-0509. — 2016 vol. 101, s. 8–17. — Wymagania systemowe: Adobe Reader. — Bibliogr. s. 16–17, Abstr.. — Publikacja dostępna online od: 2016-12-02. — YSC 2016 : 5th international Young Scientist Conference on computational science : 26–28 October 2016, Krakow, Poland. — tekst: `https://goo.gl/Nk59mc`

brak Impact Factor — punktacja (lista czasopism MNiSW, 2016): **15.000**

11. *Hypergraph grammars in non-stationary* hp*-adaptive finite element method* / Anna Paszyńska, Maciej WOŹNIAK, Andrew Lenharth, Donald Nguyen, Keshav Pingali // *Procedia Computer Science [Dokument elektroniczny]. - Czasopismo elektroniczne* ; ISSN 1877-0509. — 2016 vol. 80, s. 875–886. — Wymagania systemowe: Adobe Reader. — Bibliogr. s. 886, Abstr.. — ICCS 2016 : International Conference on Computational Science : 6–8 June 2016, San Diego, California, USA. — tekst: `http://goo.gl/VOCX0I`

brak Impact Factor — punktacja (lista czasopism MNiSW, 2016): **0.000**

12. *Hypergraph grammar based adaptive linear computational cost projection solvers for two and three dimensional modeling of brain* / Damian Goik, Marcin SIENIEK, Maciej WOŹNIAK, Anna Paszyńska, Maciej PASZYŃSKI // *Procedia Computer Science [Dokument elektroniczny]. - Czasopismo elektroniczne* ; ISSN 1877-0509. — 2014 vol. 29, s. 1002–1013. — Bibliogr. s. 1013, Abstr.. — ICCS 2014 : 14th International Conference on Computational Science : [Cairns, Australia, June 10-12, 2014]. — tekst: `http://www.sciencedirect.com/science/article/pii/S1877050914002671/pdf?md5=1ffdd9094f61e1d690a2dccfa9c1e781&pid=1-s2.0-S1877050914002671-main.pdf`

brak Impact Factor — punktacja (lista czasopism MNiSW, 2014): **10.000**

13. *Multi-frontal multi-thread direct solver with galois system for adaptive finite element method* / Anna Paszyńska, Konrad JOPEK, Maciej WOŹNIAK, Maciej PASZYŃSKI, Donald Nguyen, Andrew Lenerth, Keshav Pingali // W: *PANACM 2015 [Dokument elektroniczny]* : 1st Pan-American Congress on Computational Mechanics in conjunction with the XI Argentine congress on Computational mechanics, MECOM 2015 : 27–29 April, 2015, Buenos Aires, Argentina : proceedings / eds. Sergio R. Idelsohn, [et al.]. — Wersja do Windows. — Dane

tekstowe. — Barcelona : CIMNE, 2015. — e-ISBN: 978-84-943928-2-5. — S. 931–942. — Wymagania systemowe: Adobe Reader. — Tryb dostępu: http://congress.cimne.com/panacm2015/frontal/doc/EbookPANACM2015.pdf [2015-06-10] . — Bibliogr. s. 941–942, Abstr.. — Abstract dostępny również W: http://congress.cimne.com/PANACM2015/admin/files/fileabstract/a187.pdf

brak Impact Factor — punktacja MNiSW (2015): **15.000**

14. *Open source JAVA implementation of the parallel multi-thread alternating direction isogeometric L2 projections solver for material science simulations* — Otwarte oprogramowanie zawierające implementację w języku Java równoległego solwera wielowątkowego metody zmienno-kierunkowych izogeometrycznych L2 projekcji w zastosowaniach do symulacji inżynierii materiałowej / Grzegorz GURGUL, Maciej WOŹNIAK, Marcin ŁOŚ, Danuta SZELIGA, Maciej PASZYŃSKI // *Computer Methods in Materials Science* : quarterly / Akademia Górniczo-Hutnicza ; ISSN 1641-8581. — Tytuł poprz.: Informatyka w Technologii Materiałów. — 2017 vol. 17 no. 1, s. 1–11. — Bibliogr. s. 10–11, Abstr., Streszcz.. — KomPlasTech 2017 : XXIV Conference on Computer Methods in Materials Technology : Zakopane, Poland 15–18 January 2017. — tekst: http://www-1cmms-1agh-1edu-1pl-1atoz.wbg2.bg.agh.edu.pl/repo_file.php?f_id=568

brak Impact Factor — punktacja (lista B czasopism MNiSW, 2016): **12.000**

15. *Open source JAVA implementation of the parallel multi-thread alternating direction isogeometric L2 projections solver for material science simulations* / Grzegorz GURGUL, Maciej WOŹNIAK, Marcin ŁOŚ, Danuta SZELIGA, Maciej PASZYŃSKI // W: *KomPlasTech 2017* : XXIV international conference on Computer methods in materials technology : January 15–18, 2017, Zakopane, Poland : book of abstracts / ed. by Danuta Szeliga, Łukasz Rauch ; AGH University of Science and Technology ; Silesian University of Technology. — [Zakopane : s. n.], [2017]. — ISBN: 978-83-947091-0-5. — S. 83–84. — Bibliogr. s. 84

brak Impact Factor — punktacja MNiSW (2016): **0.000**

16. *Optimization of execution time, energy consumption and accuracy during finite element method simulations* / Maciej WOŹNIAK, Marcin ŁOŚ, Leszek SIWIK, Dariusz KRÓL, Maciej PASZYŃSKI // W: *KU KDM 2017* : tenth ACC Cyfronet AGH HPC users' conference : Zakopane 8–10 March 2017 : proceedings / ed. Kazimierz Wiatr, Jacek Kitowski, Marian Bubak. — Kraków : ACC Cyfronet AGH, 2017. — ISBN: 978-83-61433-23-1. — S. 35-36. — Bibliogr. s. 36

brak Impact Factor — brak punktacji MNiSW

17. *Parallel alternating direction preconditioner for isogeometric simulations of explicit dynamics* / Marcin ŁOŚ, Maciej WOŹNIAK, Maciej PASZYŃSKI, Lisandro Dalcin, Victor Calo // W: *PANACM 2015 [Dokument elektroniczny]* : 1st Pan-American Congress on Computational Mechanics in conjunction with the XI Argentine congress on Computational mechanics, MECOM 2015 : 27–29 April, 2015, Buenos Aires, Argentina : technical program. — Wersja do Windows. — Dane tekstowe. — Barcelona : CIMNE, 2015. — S. [1]. — Wymagania systemowe: Adobe Reader. — Tryb dostępu: http://congress.cimne.com/PANACM2015/admin/files/fileabstract/a389.pdf [2015-07-07] . — Bibliogr. s. [1]

brak Impact Factor — brak punktacji MNiSW

18. *Parallel isogeometric simulations and inversion of hazardous environmental effects during oil/gas extraction* / Marcin ŁOŚ, Maciej WOŹNIAK, Maciej PASZYŃSKI, Leszek SIWIK, Aleksander BYRSKI, Marek KISIEL-DOROHINICKI // W: *USACM thematic conference on Isogeometric analysis and meshfree methods [Dokument elektroniczny]* : [San Diego, October 10–12, 2016] : abstracts. (Pt. 2,, M-Z). — Wersja do Windows. — Dane tekstowe. — [USA : s. n.], [2016]. — S. [1]. — Wymagania systemowe: Adobe Reader. — Tryb dostępu: http://iga-mf.usacm.org/sites/default/files/IGA-MF%20Abstracts-2R.pdf [2017-01-17] . — Bibliogr. s. [1]

brak Impact Factor — brak punktacji MNiSW

19. *Quasi-optimal elimination trees for 2D grids with singularities* / A. Paszyńska, M. PASZYŃSKI, K. JOPEK, M. WOŹNIAK, D. Goik, P. GURGUL, [et al.] // *Scientific Programming* ; ISSN 1058-9244. — 2015 Article ID 303024, s. 1–18. — Bibliogr. s. 17–18

0.455 punktacja (lista A czasopism MNiSW, 2015): **11.250**

20. *Scalability of direct solver for non-stationary Cahn-Hilliard simulations with linearized time integration scheme* / M. WOŹNIAK, M. SMOŁKA, A. Cortes, M. PASZYŃSKI, R. SCHAEFER // *Procedia Computer Science [Dokument elektroniczny]. - Czasopismo elektroniczne* ; ISSN 1877-0509. — 2016 vol. 80, s. 834–844. — Wymagania systemowe: Adobe Reader. — Bibliogr. s. 843–844, Abstr.. — Publikacja dostępna online od: 2016-06-01. — ICCS 2016 : International Conference on Computational Science : 6–8 June 2016, San Diego, California, USA. — tekst: `http://goo.gl/5MDhlK`

brak Impact Factor punktacja (lista czasopism MNiSW, 2016): **0.000**

21. *Simulations of the propagation of electromagnetic waves over a human head based on projection based interpolation of MRI scan data* / Maciej WOŹNIAK, Marcin SIENIEK, Maciej PASZYŃSKI // W: *GEASC Global Engineering & Applied Science Conference ; SEDT International Symposium on Electrical, Electronic Engineering and Digital Technology ; ILSBE International Conference on Life Science and Biological Engineering [Dokument elektroniczny]* : the joint conference : December, 2015 Tokyo, Japan : conference proceedings. — Wersja do Windows. — Dane tekstowe. — [Tokyo : s. n.], [2015]. — 1 dysk Flash. — dod. ISBN: 978-986-5654-05-4 (SEDT), ISBN: 978-986-5654-0407 (ILSBE). — e-ISBN: 978-986-5654-31-3. — S. 455–462. — Wymagania systemowe: Adobe Reader. — Bibliogr. s. 462, Abstr.. — publikacja zamieszczona w GEASC

brak Impact Factor brak punktacji MNiSW

22. *Using the system of graph grammar for generation of quasi optimal element partition trees in two dimensions* — Zastosowanie systemu gramatyk grafowych do generacji quasi-optymalnych drzew podziałów siatki w dwóch wymiarach / Anna Paszyńska, Iwona Świderska, Maciej WOŹNIAK, Konrad JOPEK, Maciej PASZYŃSKI, Ewa Grabska, Andrew Lenhart, Donals Nguyen, Keshav Pingali // *Computer Methods in Materials Science* : quarterly / Akademia Górniczo-Hutnicza ; ISSN 1641-8581. — Tytuł poprz.: Informatyka w Technologii Materiałów. — 2016 vol. 16 no. 3, s. 143–155. — Bibliogr. s. 155, Abstr., Streszcz.. — tekst: `http://www-1cmms-1agh-1edu-1pl-1atoz.wbg2.bg.agh.edu.pl/repo_file.php?f_id=583`

brak Impact Factor punktacja (lista B czasopism MNiSW, 2016): **12.000**