Computer Science • 14 (4) 2013 http://dx.doi.org/10.7494/csci.2013.14.4.589

Krzysztof Kuźnik
Maciej Paszyński
Victor Calo

GRAMMAR BASED MULTI-FRONTAL SOLVER FOR ISOGEOMETRIC ANALYSIS IN 1D

Abstract *In this paper, we present a multi-frontal direct solver for one-dimensional isogeometric finite element method. The solver implementation is based on the graph grammar (GG) model. The GG model allows us to express the entire solver algorithm, including generation of frontal matrices, merging, and eliminations as a set of basic undividable tasks called graph grammar productions. Having the solver algorithm expressed as GG productions, we can find the partial order of execution and create a dependency graph, allowing for scheduling of tasks into shared memory parallel machine. We focus on the implementation of the solver with NVIDIA CUDA on the graphic processing unit (GPU). The solver has been tested for linear, quadratic, cubic, and higher-order B-splines, resulting in logarithmic scalability.*

Keywords graph grammar, direct solver, isogeometric finite element method, NVIDIA CUDA GPU

1. Introduction

The finite element method with hierarchical shape functions [6, 7] delivers higher order polynomials, but the continuity of the global approximation is only C^0 between particular mesh elements. This was the main motivation for developing the isogeometric finite element method [5], which utilizes the B-splines as basis functions and, thus, delivers C^k global continuity.

The multi-frontal direct solver is a state-of-the art algorithm for solving sparse linear systems of equations generated by finite element discretizations [9, 13]. The multi-frontal solver algorithm is a generalization of the frontal solver algorithm [15, 8].

There were several attempts at parallelization of the multi-frontal direct solver, targeting distributed memory, shared memory, or hybrid architectures. These attempts were based on either partitioning of the computational domain into subdomains with overlapping or non-overlapping sub-domains [24, 25], redistribution of the global matrix [14], or redistribution of the elimination tree into processors [21, 22, 23, 26]. There were also attempts to develop a shared-memory version of the multi-frontal solver, targeting the linux cluster nodes with multiple cores [10, 11, 12].

In this paper, we present the multi-frontal direct solver algorithm for GPU, which is a hybrid parallel machine. Our implementation has been tested on a GeForce GTX 260 device which has 24 multiprocessors with 8 CUDA cores per multiprocessor, which gives us 192 CUDA cores. The total amount of global memory is 896 megabytes.

The graph grammar based multi-frontal solver for isogeometric computations presented in this paper is the generalization of the GG based solver already developed for one-dimensional finite difference simulation [17]. The graph grammar based analysis of operations concurrency in the standard finite element method has been explored for C^0 continuous elements, with focus on the generation of two dimensional computational grids with rectangular, triangular, and mixed finite elements [19, 20, 18].

The graph grammar based model allows us to investigate if concurrency is hidden within the algorithm. It is done by analyzing the partial order of execution of the graph grammar productions, namely the basic undividable tasks, and the identification of sets of productions that can be executed concurrently [23].

In this paper, we generalize the idea of the graph grammar based solver into one dimensional B-spline-based finite element method, delivering C^k global continuity of the solution. The methodology derived here implies logarithimic scalability of the parallel multi-frontal solver algorithm. The methodology has been implemented and tested on NVIDIA CUDA GPU, providing logarithmic execution time for linear, quadratic, cubic, quartic, and quintic B-splines.

The motivation for using GPU cards for one dimensional isogeometric finite element method is as follows: first, the isogeometric L2 projection in two or three dimensions can be expressed as a sequence of a solution of one dimensional problems with multiple right hand sides [4]. The L2 projection can be applied for the solution of the non-stationary time dependent problems with the Forward Euler scheme.

Second, the solver can be utilized for an efficient solution of non-stationary time dependent problems when we need to perform several iterations, with either the Forward Euler, Backward Euler or Crank-Nicolson methods. For all of these schemes, the problem can be formulated in such a way that only the boundary conditions and right hand side changes when we switch from one iteration to the other. This implies that the LU factorization of the matrix can be performed only once, for the initial time step. The boundary conditions are enforced just at the root of the elimination tree, and the change of the right hand side implies the necessity of one forward elimination followed by backward substitutions. The LU factorization, although expensive, is performed only once. In the following time steps, we can perform a single forward and backward substitution. The time advantage of the usage of the GPU with respect to CPU with MUMPS solver will allow for the solution of the non-stationary problem one order of magnitude faster.

2. B-splines based Finite Element Method

We focus on one dimensional elliptic problem with mixed Dirichlet and Neumann boundary conditions.

$$-\frac{d}{dx}\left(\kappa\left(x\right)\frac{du\left(x\right)}{dx}\right)=f\left(x\right) \tag{1}$$

$$u\left(0\right)=0 \tag{2}$$

$$\kappa\left(1\right)\frac{du\left(1\right)}{dx}=\gamma \tag{3}$$

The weak form of (1-3) is obtained by using L^2 inner product with test function v and integrating by parts, that is,

$$\text{Find } u\in V=\left\{u\in H^1\left(0,1\right):u\left(0\right)=0\right\} \text{ such that} \tag{4}$$

$$b\left(v,u\right)=l\left(v\right),\forall v\in V \tag{5}$$

$$\text{where } b\left(u,v\right)=\int_0^1\kappa\left(x\right)\frac{dv\left(x\right)}{dx}\frac{du\left(x\right)}{dx}dx \tag{6}$$

$$l\left(v\right)=\int_0^1 f\left(x\right)v\left(x\right)dx+\gamma v\left(1\right) \tag{7}$$

For isogeometric analysis, we approximate the solution with B-spline basis functions

$$u\left(x\right)\approx\sum_i N_{i,p}\left(x\right)a_i \qquad v\in\left\{N_{j,p}\right\}_j \tag{8}$$

where

$$N_{i,0}\left(\xi\right)=I_{[\xi_i,\xi_{i+1}]} \tag{9}$$

$$N_{i,p}\left(\xi\right)=\frac{\xi-\xi_i}{x_{i+p}-\xi_i}N_{i,p-1}\left(\xi\right)+\frac{\xi_{i+p+1}-\xi}{x_{i+p+1}-\xi_{i+1}}N_{i+1,p-1}\left(\xi\right) \tag{10}$$

where $I_{[\xi_i,\xi_{i+1}]}$ is the identity function over the interval $[\xi_i, \xi_{i+1}]$.

Substituting these definitions in the weak form allows us to obtain the discrete weak formulation, which is stated as

$$\sum_i b\left(N_{j,p}\left(x\right), N_{i,p}\left(x\right)\right) a_i = l\left(N_{j,p}\left(x\right)\right), \forall j \tag{11}$$

To evaluate the integrals implied in (11), numerical quadrature rules are the method of choice due to their flexibility and efficiency. In this application, we use Gaussian rules to calculate (11). For each row in (11), the integral is restricted to the support of $N_{j,p}$. This support is broken into elements, defined as segments where $N_{i,p}$ and $N_{j,p}$ are simple polynomials. The integral $b\left(N_{j,p}, N_{i,p}\right)$ restricted to an element e is denoted the element stiffness matrix, while the restriction of $l\left(N_{j,p}\right)$ to the element is denoted element force vector.

3. Grammar productions expressing the generation of element local matrices

The computational domain $\Omega = [0,1]$ is partitioned into N finite elements $\Omega = \cup_{k=1,\ldots,N}[\xi_k, \xi_{k+1}] = \cup_{i=1,\ldots,N}[\frac{k-1}{N}, \frac{k}{N}]$. We begin with the generation of element local matrices called the frontal matrices. Each element $e_k = [\xi_k, \xi_{k+1}]$ frontal matrix is generated by computing the integrals $b\left(N_{j,p}\left(x\right), N_{i,p}\left(x\right)\right)$ for B-splines restricted over the element. For linear B-splines, there are two linear B-splines $p = 1$ $N_{k,1}$ and $N_{k-1,1}$ having support over the element e_k. Thus, the element frontal matrix consists in $2 \times 2 = 4$ entries, as it is illustrated in Figure 1 and in Table 1. For quadratic B-splines $p = 2$, there are three functions $N_{k,2}$, $N_{k-1,2}$, and $N_{k-2,2}$ with support over the element e_k. The element frontal matrix has $3 \times 3 = 9$ entries, see Figure 2 and Table 2. In general, for p order B-splines, we have $N_{k,p}, \ldots, N_{k-m,p}$ functions with support over element e_k and the element frontal matrix has $(p+1)^2$ entries.

Table 1

For linear B-splines there are 2 functions over each element, so there are $2 \times 2 = 4$ elemental matrix entries.

$b\left(N_{k-1,1}\left(x\right), N_{k-1,1}\left(x\right)\right)$	$b\left(N_{k-1,1}\left(x\right), N_{k,1}\left(x\right)\right)$
$b\left(N_{k,1}\left(x\right), N_{k-1,1}\left(x\right)\right)$	$b\left(N_{k,1}\left(x\right), N_{k,1}\left(x\right)\right)$

Each matrix entry is an integral that must be computed by using Gaussian quadrature rules, see (12–13).

$$b\left(N_{j,p}\left(x\right), N_{i,p}\left(x\right)\right) = \int_0^1 A\left(x\right) \frac{dN_{j,p}\left(x\right)}{dx} \frac{dN_{i,p}\left(x\right)}{dx} dx + \beta N_{j,p}\left(1\right) N_{i,p}\left(1\right) =$$
$$\sum_G w_G A\left(x_G\right) \frac{dN_{j,p}\left(x_G\right)}{dx} \frac{dN_{i,p}\left(x_G\right)}{dx} + \beta N_{j,p}\left(1\right) N_{i,p}\left(1\right) \tag{12}$$

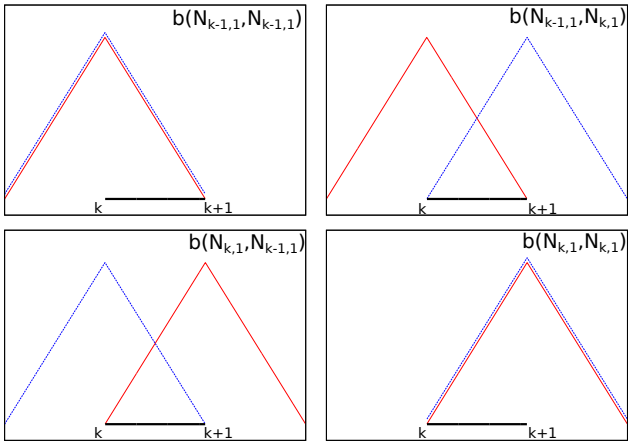

Figure 1. Basis functions interactions at element contribute to the element matrix for linear B-splines.

$$l\left(N_{j,p}\right)=\int_0^1 f\left(x\right)N_{j,p}\left(x\right)dx+\gamma N_{j,p}\left(1\right)=\sum_G w_G f\left(x_G\right)N_{j,p}\left(x_G\right)+\gamma N_{j,p}\left(1\right) \tag{13}$$

To compute the formulae (12-13), we need to compute the values of the B-spline functions and their derivatives at Gauss points x_G and at the end points. The process of computation of the value of a B-spline at given Gauss point can be decomposed into basic tasks. Similarly, the process of evaluating the functions and their derivatives at the end point $x=1$ can be divided into grammar productions. This last process is not described, since it mirrors the analysis performed at Gauss points.

Table 2

For quadratic B-splines there are 3 functions over each element, so there are $3\times 3=9$ elemental matrix entries.

$b\left(N_{k-2,2}\left(x\right),N_{k-2,2}\left(x\right)\right)$	$b\left(N_{k-2,2}\left(x\right),N_{k-1,2}\left(x\right)\right)$	$b\left(N_{k-2,2}\left(x\right),N_{k,2}\left(x\right)\right)$
$b\left(N_{k-1,2}\left(x\right),N_{k-2,2}\left(x\right)\right)$	$b\left(N_{k-1,2}\left(x\right),N_{k-1,2}\left(x\right)\right)$	$b\left(N_{k-1,2}\left(x\right),N_{k,2}\left(x\right)\right)$
$b\left(N_{k,2}\left(x\right),N_{k-2,2}\left(x\right)\right)$	$b\left(N_{k,2}\left(x\right),N_{k-1,2}\left(x\right)\right)$	$b\left(N_{k,2}\left(x\right),N_{k,2}\left(x\right)\right)$

For linear B-splines, the computation of the value of the B-spline at given Gauss point over an element e_k is straightforward. Linear B-splines have support over two consecutive elements, see Figure 3. We need to compute the value of $N_{k,1}\left(x_G\right)$ and the value of $N_{k-1,1}\left(x_G\right)$ The derivatives of the linear B-splines illustrated in Figure 4 also require computation of the value of $N'_{k,1}\left(x_G\right)$ and the value of $N'_{k-1,1}\left(x_G\right)$.

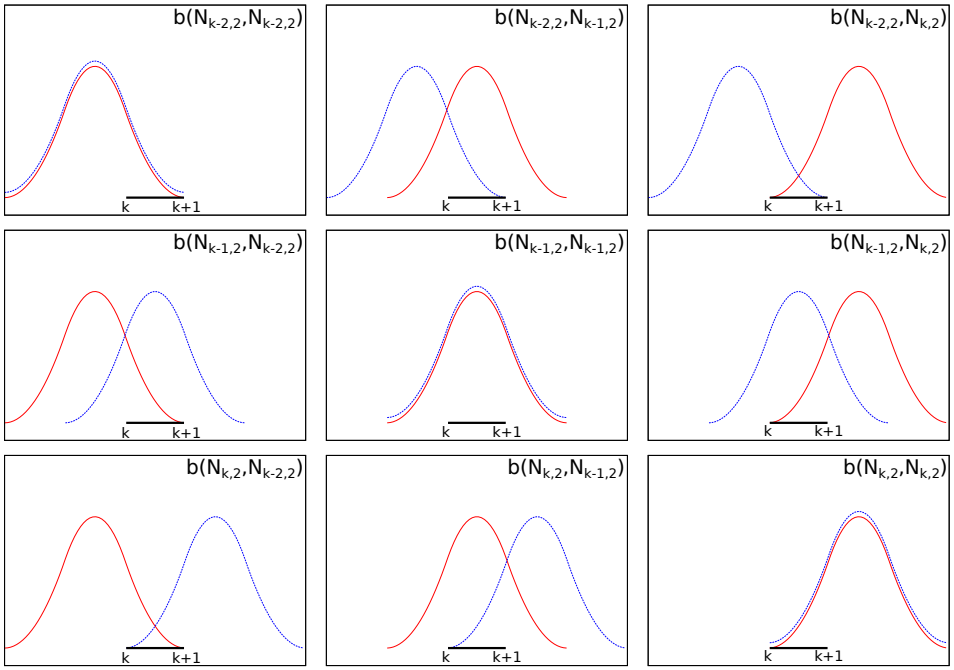

Figure 2. Basis functions interactions at element contribute to the element matrix for quadratic B-splines.

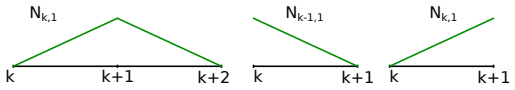

Figure 3. Basic tasks for computing of the value of linear B-spline at Gauss point x_G.

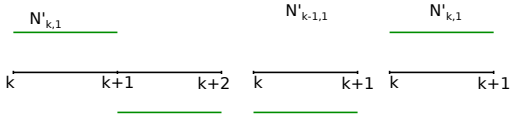

Figure 4. Basic tasks for computing of the value of derivative of linear B-spline at Gauss point x_G.

Note that a quadratic B-spline can be represented as the summation of two consecutive piecewise linear B-splines each multiplied by a linear function with identical support to them, see Figure 5. Thus, the support of the quadratic B-splines spreads over three elements. That is, to compute the contribution to the frontal matrix, we need to compute $N_{k,2}(x_G)$ with $N_{k-1,2}(x_G)$ and $N_{k-2,1}(x_G)$ at the Gauss points. These functions are obtained by multiplicating previously-computed values of $N_{k-1,1}(x_G)$, and $N_{k,1}(x_G)$ by the appropriate linear functions with extended support. The pro-

cedure is illustrated in Figure 5. Derivatives of quadratic B-splines require a little more effort, since we need to utilize previously-computed values of linear B-splines and their derivatives, see Figure 6.

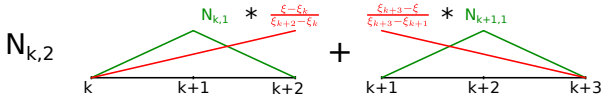

Figure 5. Basic tasks for computing of the value of quadratic B-spline at Gauss point x_G.

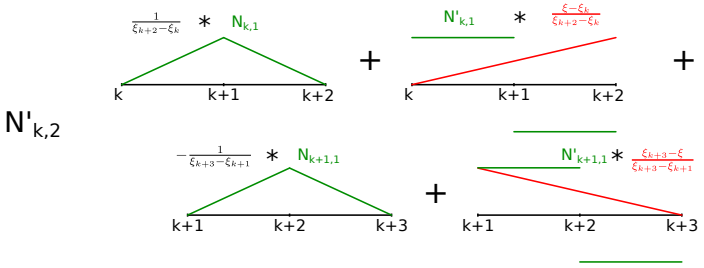

Figure 6. Basic tasks for computing of the value of derivative of quadratic B-spline at Gauss point x_G.

The procedure can be easily generalized for higher order B-splines, since it utilizes the recursive structure of the B-splines formulae (9-10). The procedure for cubic B-splines and their derivatives is illustrated in Figures 7 and 8.

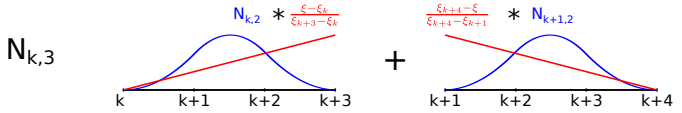

Figure 7. Basic tasks for computing of the value of cubic B-spline at Gauss point x_G.

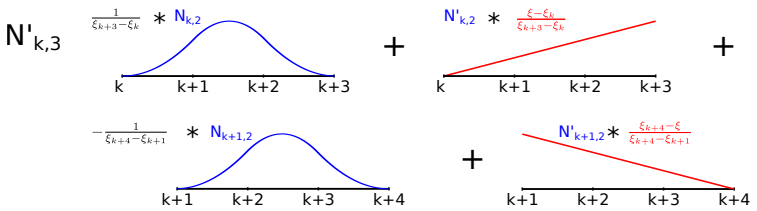

Figure 8. Basic tasks for computing of the value of derivative of cubic B-spline at Gauss point x_G.

4. Scheduler for concurrent generation of element local matrices

The generation of element frontal matrices for B-splines of order p involve computations of values of B-splines and their derivatives at Gauss points followed by the collection of these values to evaluate the matrix entries. The values of p-th order B-splines and their derivatives have been decomposed into atomic operations in section 3. These operations reduce to a sequence of computations of lower order B-splines and their derivatives, starting from the linear one. The formulation of matrix entries is expressed just by the multiplication of values of B-splines and their derivatives by values of the material data function $A(x_G)$ and Gauss weights.

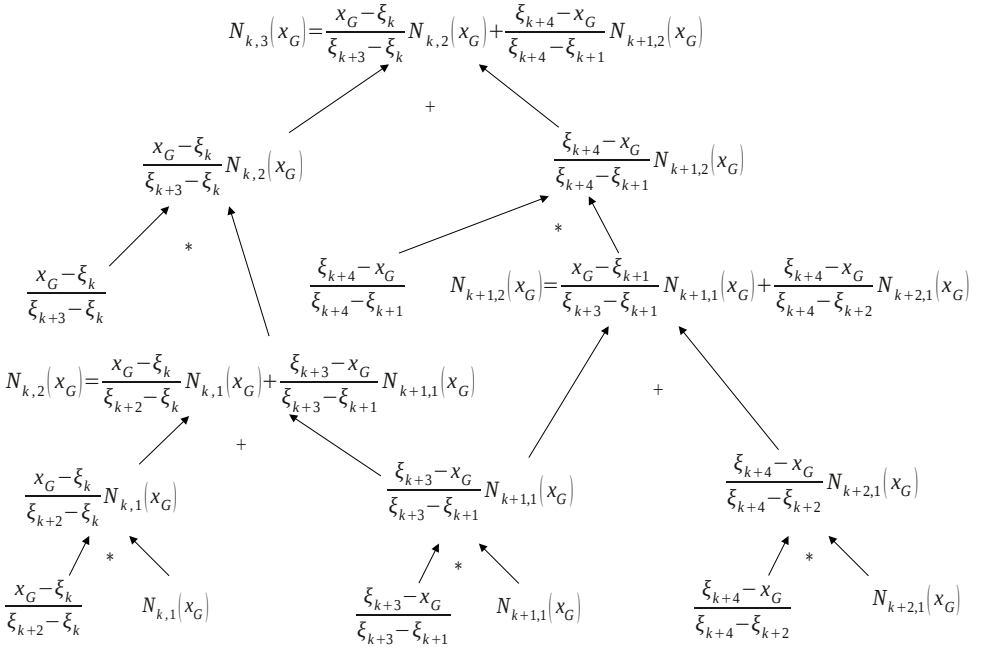

Figure 9. Scheduling of tasks computing cubic B-spline values at Gaussian quadrature points.

For example, the process of generating the element local matrices for cubic B-splines is summarized in the following (and sketched in Figure 9). We start by computing the values of the linear B-splines and their derivatives at the Gassian quadrature points. The tasks responsible for these computations are named **Nk,1**, **Nk-1,1** and **N'k,1** and **N'k-1,1**. These tasks are depicted in Figure 9 (in red) and can be executed concurrently. In the next step, we utilize previously-computed values of linear B-splines and their derivatives to compute the values of the quadratic B-splines and their derivatives. These tasks are named **Nk,2**, **Nk-1,2**, **Nk-2,2**, as well as **N'k,2**, **N'k-1,2** and **N'k-2,2**. These tasks are shown in Figure 9 (in green) and can be exe-

cuted concurrently after all red tasks are finished. In the third step, we utilize the previously-computed values of the quadratic B-splines and their derivatives to finally compute the values of the cubic B-splines and their derivatives. The tasks computing values of the cubic B-splines are named **Nk,3**, **Nk-1,3**, **Nk-2,3** and **Nk-3,3** while the tasks computing values of the derivatives of cubic B-splines are named **N'k,3**, **N'k-1,3**, **N'k-2,3** and **N'k-3,3**. These tasks shown in Figure 9 (in blue), can be executed concurrently after all green tasks are finished. Finally, we compute the values of matrix entries using values and derivatives of cubic B-splines. These tasks are named **b(Ni,3, Nj,3)** where $i, j = k, \ldots, k-3$. They are denoted in Figure 9 in gray, and as before, all of these tasks can be executed concurrently after all blue tasks are finished.

5. Grammar productions expressing the multi-frontal solver algorithm

In this section, we introduce the multi-frontal solver algorithm on the simplified version of (1-3) one dimensional heat transfer problem. That is

$$\frac{d^2u}{dx^2} = 0 \quad \forall x \in [0,1] \quad u(0) = 0 \quad \frac{du(1)}{dx} = 1 \tag{14}$$

According to the general formulation derived in (1-3), we have set $A(x) = 1$, $f(x) = 0$ and $\gamma = 1$. The weak formulation can be obtained by substituting to the general scheme and (4-7). To exemplify how the multi-frontal solver works, we apply this solution methodology to a six-element discretization where the polynomial order is linear in all of them. That is, the problem is solved with linear B-splines. We can derive the global matrix, by computing and collecting all entries of (12–13).

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -h \end{bmatrix} \tag{15}$$

where h is the element's diameter. We have enforced the Dirichlet boundary condition $u_0 = 0$ in the first row and the Neumann boundary condition $\frac{u_n - u_{n-1}}{h} = 1$ in the last row. The element frontal matrices are obtained by either computing the entries (12-13) separately, with integrals projected onto particular elements, or by partitioning the global system (15) into several sub-matrices.

$$\begin{bmatrix} 1 & 0 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$
$$\begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} u_4 \\ u_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} u_5 \\ u_6 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} u_6 \\ u_7 \end{bmatrix} = \begin{bmatrix} 0 \\ -h \end{bmatrix} \tag{16}$$

Notice that the local systems (16) are not equivalent to the global system (15) unless we assemble the system. The multi-frontal solver algorithm merges the first and second, third and fourth, and fifth and sixth matrices to obtain

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{17}$$

$$\begin{bmatrix} -1 & 1 & 0 \\ 1 & -2 & 1 \\ 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} u_3 \\ u_4 \\ u_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{18}$$

$$\begin{bmatrix} -1 & 1 & 0 \\ 1 & -2 & 1 \\ 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} u_5 \\ u_6 \\ u_7 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -h \end{bmatrix} \tag{19}$$

Notice that only the central row is fully assembled for all matrices; in general, the first and third rows are not fully assembled yet. The only exceptions are the first system, which has first row fully assembled, and the last system, which has the last row fully assembled. The multi-frontal solver reorders the system to place the fully-assembled central row at the beginning. We treat all systems similarly, making no distinction for the first and last systems. This allows us to apply the same taks to all frontal matrices. We will eliminate the first and the last row in the final step. The resulting reordered systems follow:

$$\begin{bmatrix} -2 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} u_2 \\ u_1 \\ u_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{20}$$

$$\begin{bmatrix} -2 & 1 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} u_4 \\ u_3 \\ u_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{21}$$

$$\begin{bmatrix} -2 & 1 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} u_6 \\ u_5 \\ u_7 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -h \end{bmatrix} \tag{22}$$

At this point, the multi-frontal solver algorithm performs the elimination of the first fully-assembled row. The first row is subtracted from the second and third rows. We can subtract the fully-assembled row from rows that are not fully assembled, because the subtraction and additions are interchangeable. That is, we subtract the fully-assembled row at this point, and in the following step, we add the remaining part of the non-fully-assembled row. The resulting systems after partial elimination are:

$$\begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & 1 & 0 \\ 0 & \frac{1}{2} & -\frac{1}{2} \end{bmatrix} \begin{bmatrix} u_2 \\ u_1 \\ u_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{23}$$

$$\begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & -\frac{1}{2} & \frac{1}{2} \\ 0 & \frac{1}{2} & -\frac{1}{2} \end{bmatrix} \begin{bmatrix} u_4 \\ u_3 \\ u_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{24}$$

$$\begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & -\frac{1}{2} & \frac{1}{2} \\ 0 & \frac{1}{2} & -\frac{1}{2} \end{bmatrix} \begin{bmatrix} u_6 \\ u_5 \\ u_7 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -h \end{bmatrix} \tag{25}$$

Now, we focus on the right bottom 2×2 sub-matrices, the part that is still missing contributions from neighboring elements. These sub-matrices are called the Schur complements. The multi-frontal solver algorithm merges the first and the second Schur complement matrices to obtain a new 3×3 system

$$\begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & -1 & \frac{1}{2} \\ 0 & \frac{1}{2} & -\frac{1}{2} \end{bmatrix} \begin{bmatrix} u_1 \\ u_3 \\ u_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{26}$$

Again, in general, only the central row is fully assembled at this level. We reorder to put the fully-assembled row at the beginning and obtain

$$\begin{bmatrix} -1 & \frac{1}{2} & \frac{1}{2} \\ 0 & 1 & 0 \\ \frac{1}{2} & 0 & -\frac{1}{2} \end{bmatrix} \begin{bmatrix} u_3 \\ u_1 \\ u_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{27}$$

We can eliminate the fully-assembled row by subtracting it from the second and third rows (which are not fully assembled as of yet) like we did before. This results in

$$\begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & 1 & 0 \\ 0 & \frac{1}{4} & -\frac{1}{4} \end{bmatrix} \begin{bmatrix} u_3 \\ u_1 \\ u_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{28}$$

Again, the right bottom 2×2 sub-matrix is our new Schur complement. Finally, the multi-frontal solver algorithm merges the last Schur complement with the third Schur complement to get the root system

$$\begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{4} & -\frac{3}{4} & \frac{1}{2} \\ 0 & \frac{1}{2} & -\frac{1}{2} \end{bmatrix} \begin{bmatrix} u_1 \\ u_5 \\ u_7 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -h \end{bmatrix} \tag{29}$$

The root problem is fully assembled, and we can perform the full forward elimination followed by backward substitution.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -\frac{2}{3} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_5 \\ u_7 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 6h \end{bmatrix} \tag{30}$$

We get the solution with $h = \frac{1}{6}$

$$u_7 = 1 \tag{31}$$

$$u_5 = \frac{2}{3}u_7 = \frac{2}{3} \tag{32}$$

$$u_1 = 0 \tag{33}$$

and we finally proceed with backward substitutions at son nodes

$$u_3 = \frac{1}{2}u_1 + \frac{1}{2}u_5 = \frac{1}{3} \tag{34}$$

$$u_2 = \frac{1}{2}u_1 + \frac{1}{2}u_3 = \frac{1}{6} \tag{35}$$

$$u_4 = \frac{1}{2}u_3 + \frac{1}{2}u_5 = \frac{1}{2} \tag{36}$$

$$u_6 = \frac{1}{2}u_5 + \frac{1}{2}u_7 = \frac{5}{6} \tag{37}$$

This solution strategy can be repeated for higher-order B-spline discretizations of arbitrary continuity at element interfaces. The process is applied in a similar fashion. All elemental contributions are computed using the scheduler described in section 4. These element contributions will be merged with neighboring elements. All rows that are fully assembled will be placed at the top of the partial matrix and eliminated. The resulting Schur complements will be assembled at the next level, after all sub-matrices in this level are partially eliminated. This procedure is applied recursively until the root system is solved and recursive backward substitution is applied. A scheduler for these tasks is described in the next section.

6. Scheduler for the solver execution

The multi-frontal solver algorithm discussed in the previous section for linear B-splines is summarized in Figure 10 (for a problem with eight elements in the domain). Each element frontal matrix has two rows related to two linear B-splines having non-zero support over an element. Two Schur complements are merged into new systems, which is performed by tasks named **Pm2**. Next, fully-assembled central rows are eliminated, and this is done by tasks named **Pe1**. The process of merging Schur complements and eliminating central rows is repeated until we reach the root of the elimination tree, where the system is fully assembled and can be solved. This is done by a task named **Ps**. The root problem solution is followed by backward substitutions. Notice that there are several tasks working on each level of the assembly tree; in particular, at the leaf-nodes level, there are N tasks where N is the number of unknowns and the number of tasks decreases logarithmically down to a single task at root of the tree. All tasks from each level of the assembly tree can be executed concurrently. Thus, the total execution time of the solver is $log\,(N)$. This is because the number of levels in the assembly tree is equal to the logarithm of the number of elements,

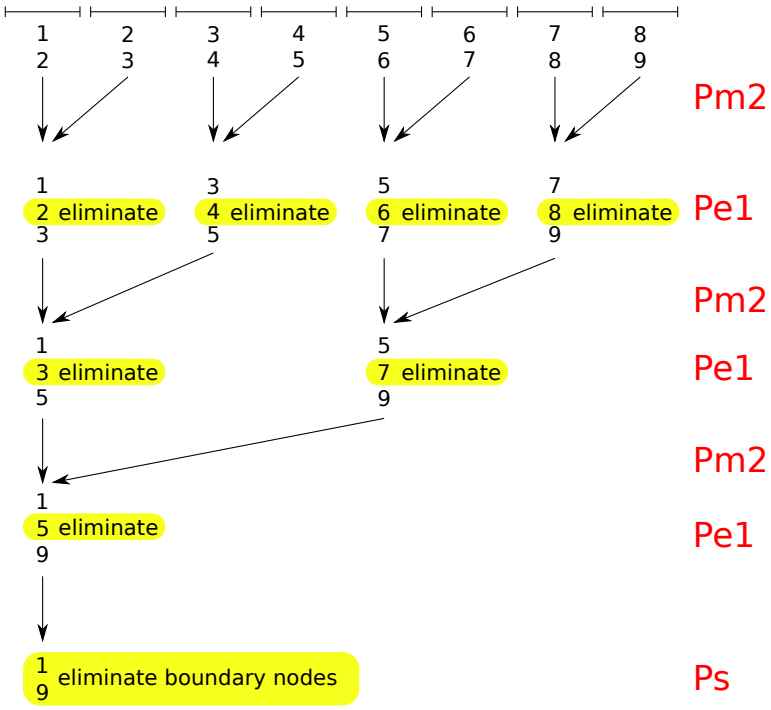

Figure 10. Scheduling of tasks for multi-frontal solver execution for linear B-splines.

which is proportional to the number of degrees of freedom. Moreover, we perform $O(1)$ operation at each node of the assembly tree.

For quadratic B-splines, the situation is little different (as illustrated in Figure 11). The element frontal matrices have three rows in this case, since there are three B-splines with support over each element. Thus, in the first step of the algorithm, the solver executes several **Pm3** tasks responsible for merging three element frontal matrices. This is followed by the execution of several **Pe1** tasks responsible for elimination of 1 central fully-assembled row. In the following steps, the algorithm merges two Schur complements from son nodes (tasks **Pm2**) and eliminates two fully-assembled rows (tasks **Pe2**). At the root, there are four fully-assembled rows, and the fully system can be solved by **Ps** task followed by backward substitution.

In general, when we utilize B-splines of order p with continuity $p-1$, we need to first merge $p+1$ frontal matrices, and we can eliminate one central row. In the following steps, we merge two Schur complements and eliminate p central fully-assembled rows. Finally, we get the root problem of size $2p$ that is solved and followed by backward substitution. The procedure is also illustrated for cubic B-splines in Figure 12. The computational complexity of the solver for order p is $p^2 log\,(N/p)$. This is because the number of levels in the assembly tree is equal to the logarithm of the number of

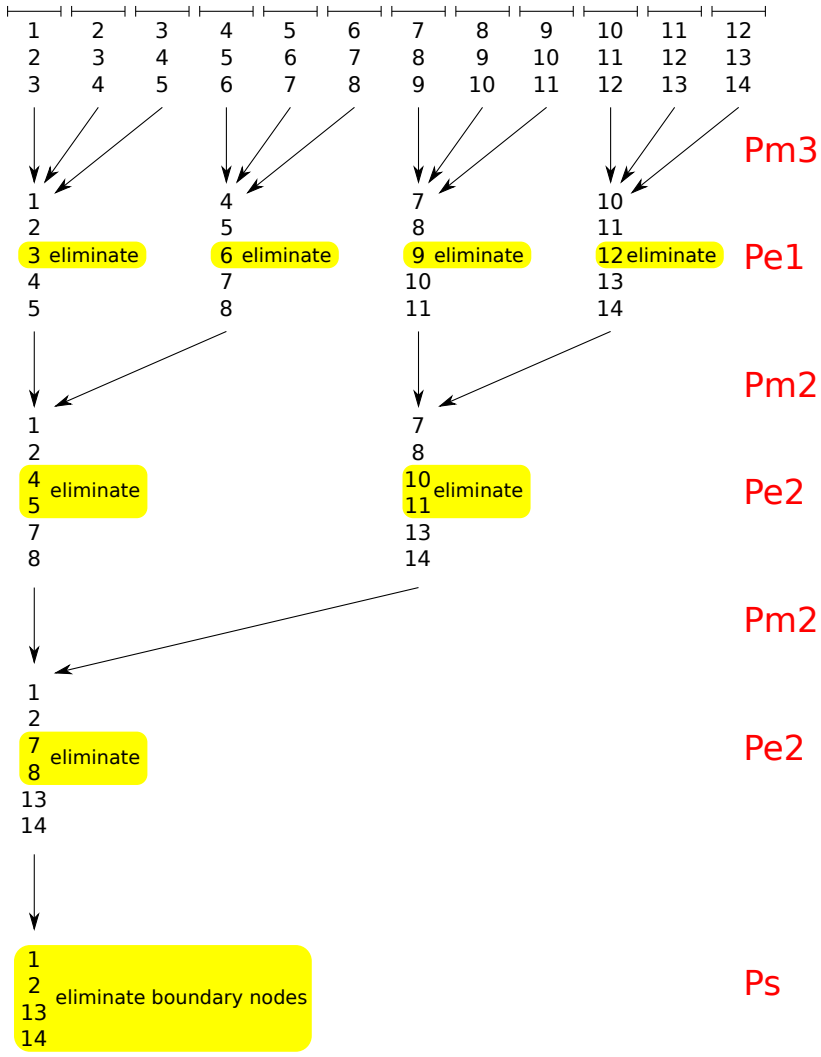

Figure 11. Scheduling of tasks for multi-frontal solver execution for quadratic B-splines.

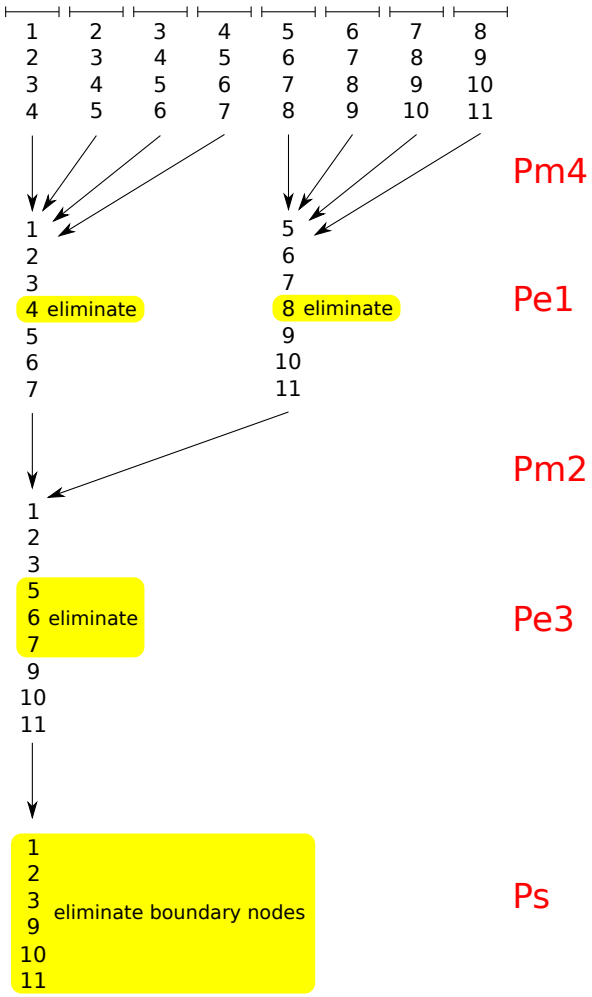

Figure 12. Scheduling of tasks for multi-frontal solver execution for cubic B-splines.

elements, which is equal to the number of degrees of freedom divided by the polynomial order p. Moreover, the size of each frontal matrix is $O(p)$, and we need to subtract p rows from the following rows (which can be done in $O(p^2)$).

7. Numerical results

We conclude the paper with numerical results obtained with grammar-based implementation of the derived methodology on NVIDIA CUDA GPU. The numerical tests were performed on GeForce GTX 260 graphic card with 24 multiprocessors.

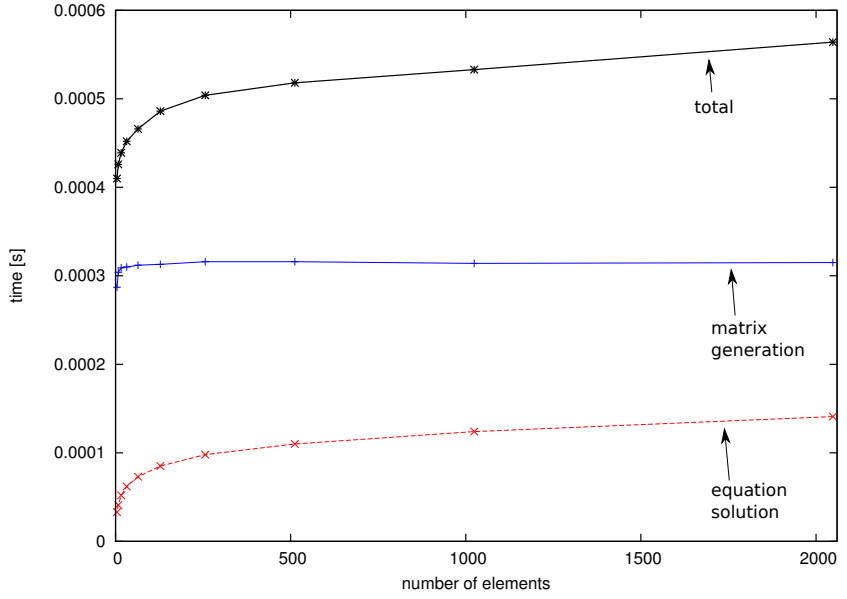

Figure 13. Execution time measured on GPU for linear B-splines.

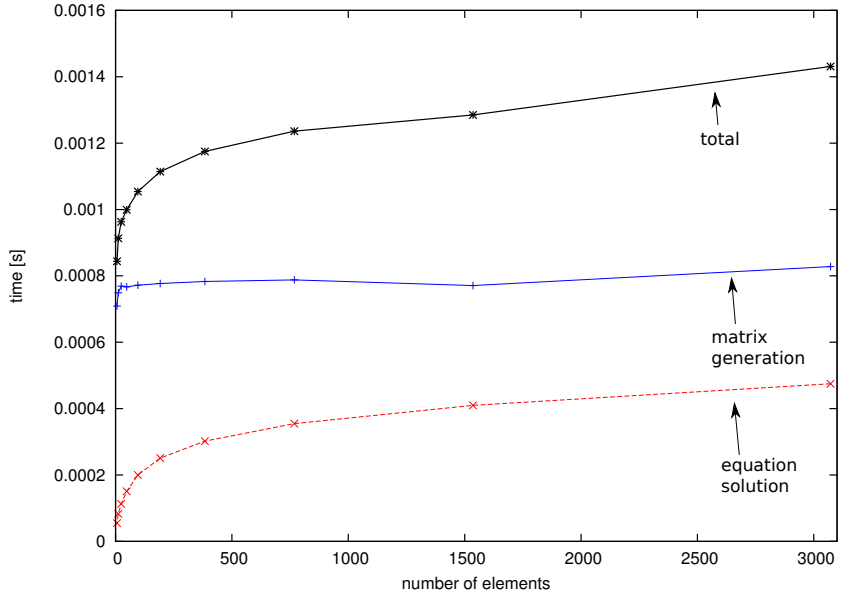

Figure 14. Execution time measured on GPU for quadratic B-splines.

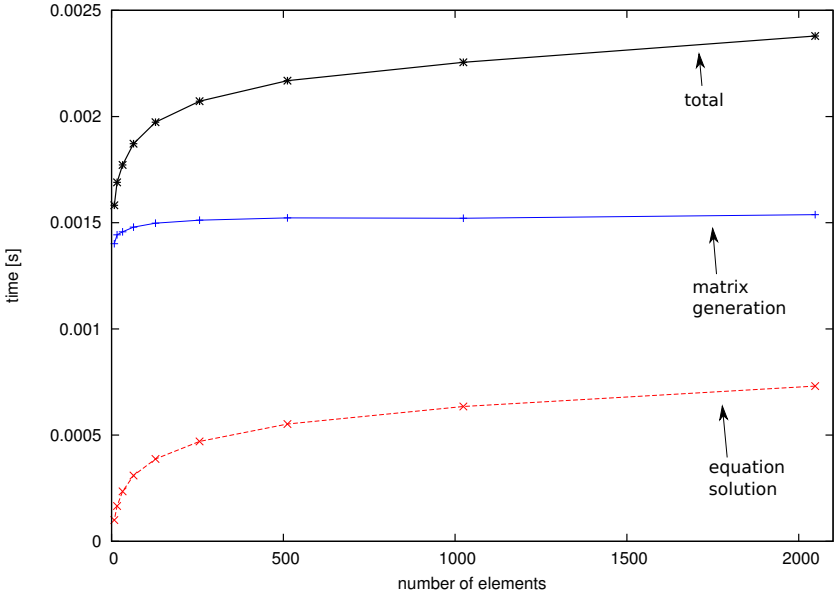

Figure 15. Execution time measured on GPU for cubic B-splines.

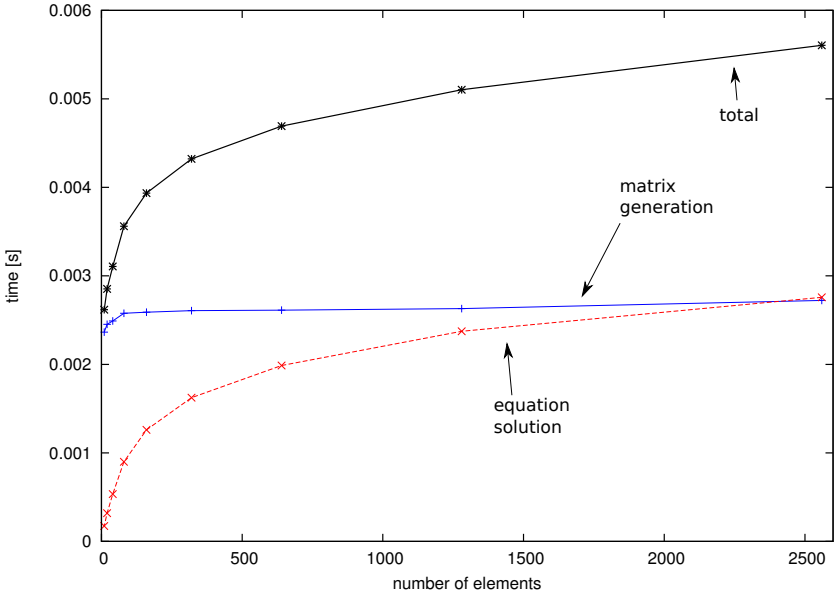

Figure 16. Execution time measured on GPU for fourth order B-splines.

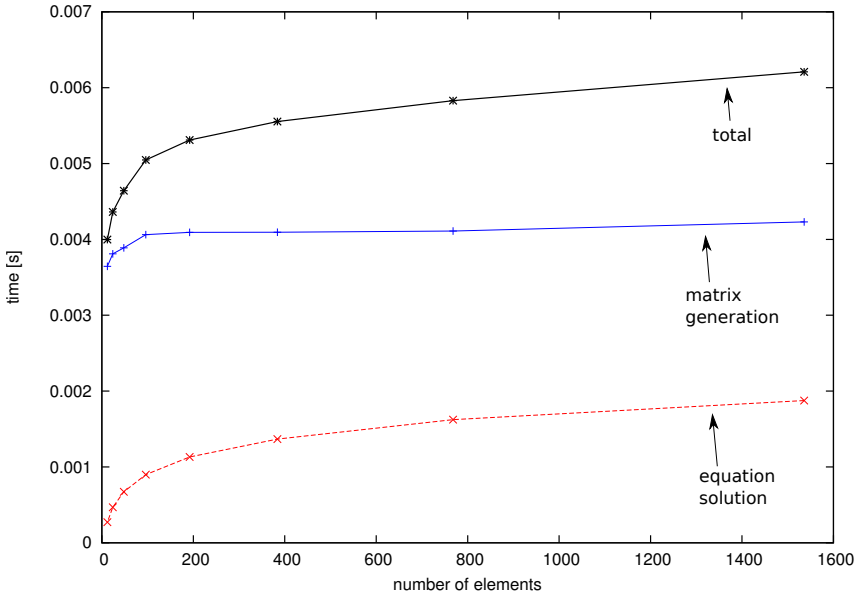

Figure 17. Execution time measured on GPU for fifth order B-splines.

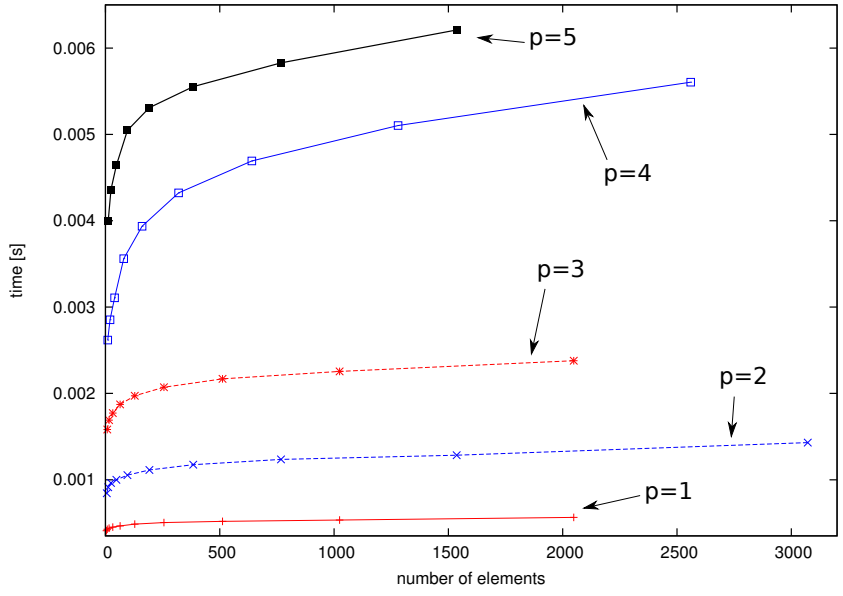

Figure 18. Comparison of GPU execution times for different orders of B-splines.

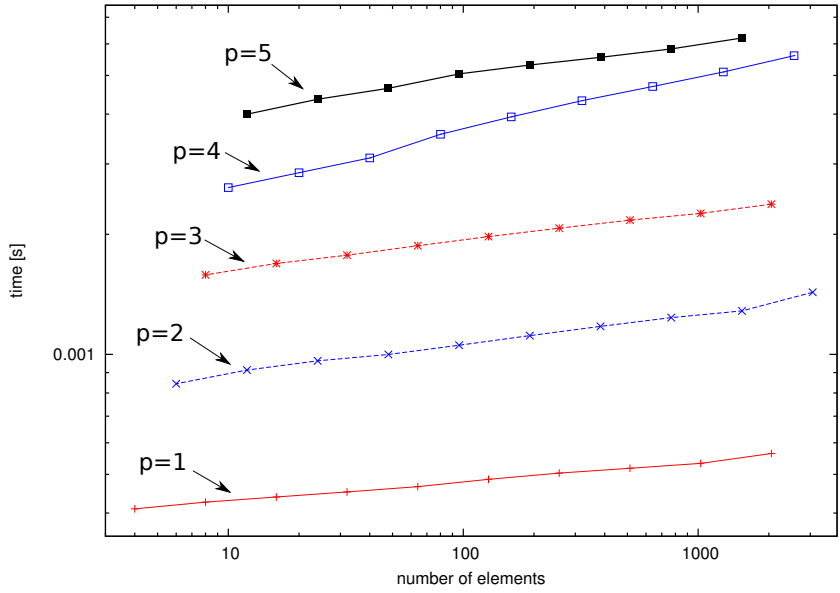

Figure 19. Comparison of GPU execution times for different orders of B-splines on log-log scales.

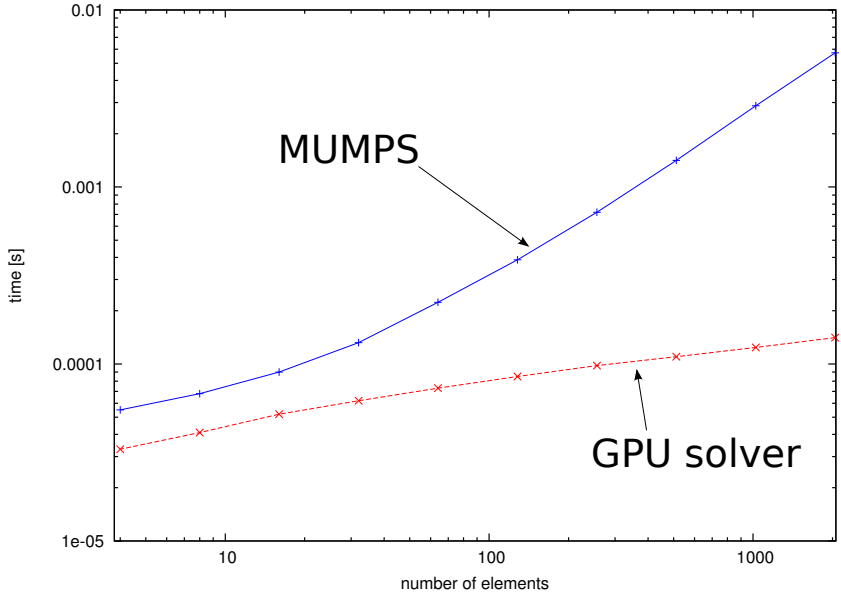

Figure 20. Comparison of GPU execution time with MUMPS solver CPU execution time for linear B-splines.

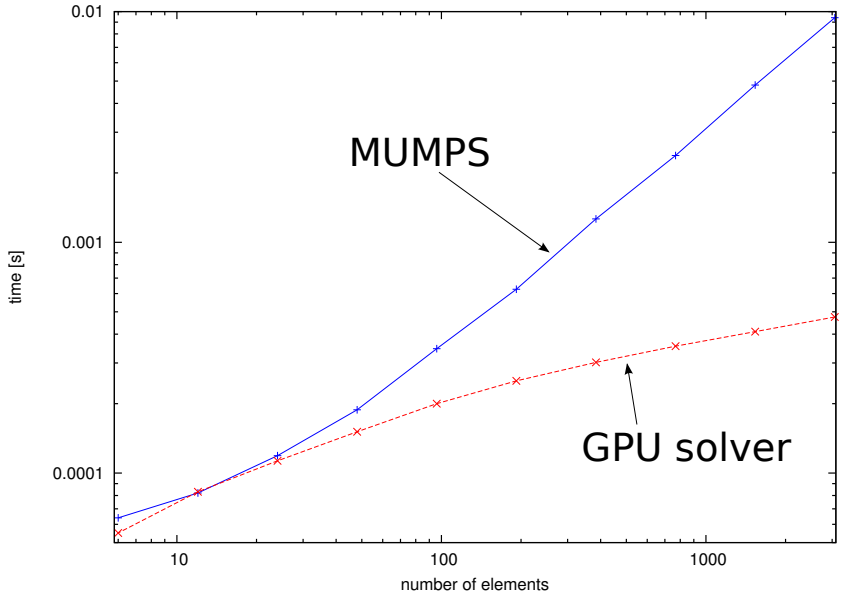

Figure 21. Comparison of GPU execution time with MUMPS solver CPU execution time for quadratic B-splines.

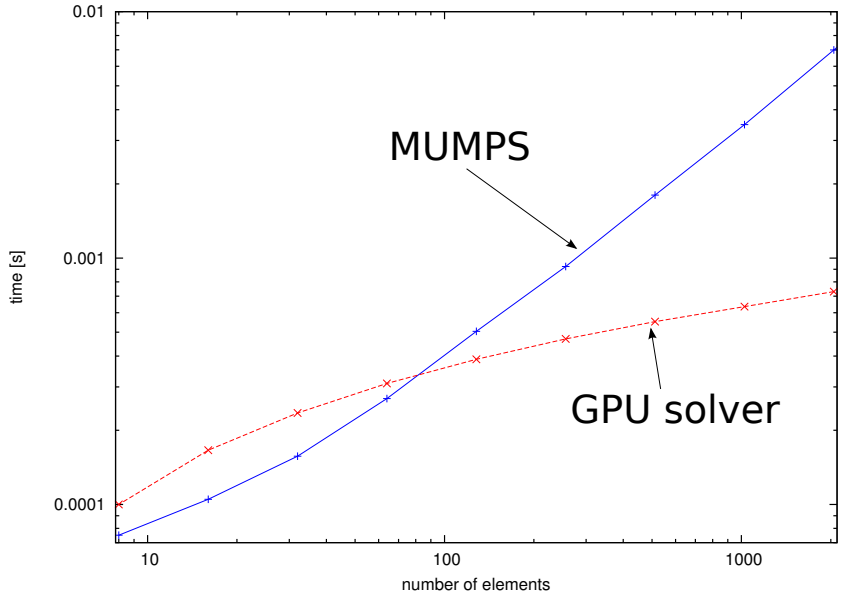

Figure 22. Comparison of GPU execution time with MUMPS solver CPU execution time for cubic B-splines.

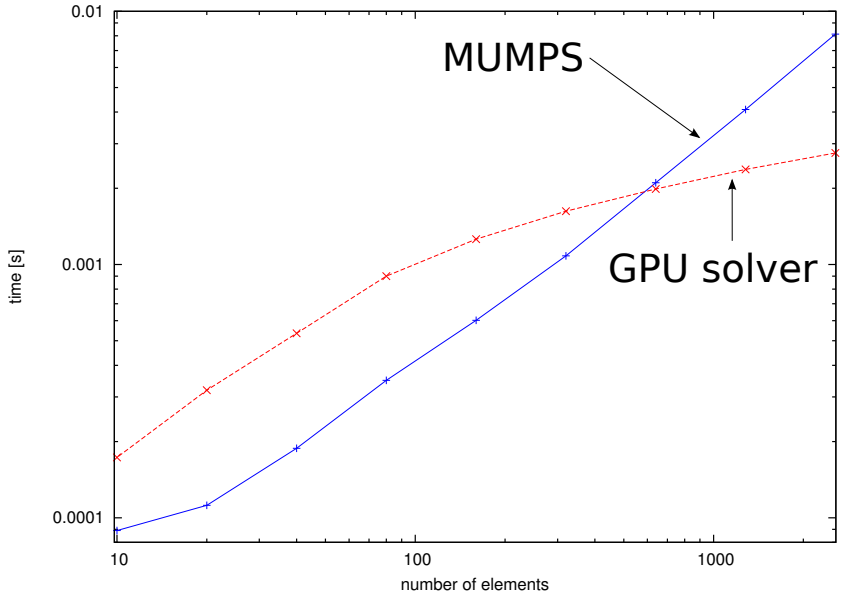

Figure 23. Comparison of GPU execution time with MUMPS solver CPU execution time for quartic B-splines.

Each equipped with 8 cores. The total number of cores is equal to 192. The global memory on graphic card was 896MB.

The numerical experiments were performed for linear, quadratic, cubic and quartic, and quintic B-splines (as illustrated in Figures 13–17). We report the execution time spent on local matrix generation and execution of the multi-frontal solver. The comparison of execution time for different orders of B-splines is summarized in Figure 18 and 19 with log-log scale.

The numerical experiments were performed for linear, quadratic, cubic, quartic, and quintic B-splines (as illustrated in Figures 13–17). We report the execution time spent on local matrix generation and execution of the multi-frontal solver. The comparison of execution time for different orders of B-splines is summarized in Figure 18 and 19 with log-log scale.

Our implementation is fast and scalable, attaining the expected logarithmic scaling predicted by the theory. However, there are a few issues that need to be discussed.

1. As the reader may notice, in most cases (all but quartics) frontal matrices generation takes twice as long to solve the equation. The reason for this is that the multi-frontal solver was quite well-optimized for global memory access. Almost all global memory reads and writes are coalesced. After the data is read, it is stored in shared memory where computations are very fast. After processing in shared memory, data is stored back in global memory in such a way that the next read will also be coalesced. This way, we minimize memory transactions which

gives us this boost. We could not achieve this level of uniform access to global memory in both steps. We chose memory layout which is better for multi-frontal solver; but for generating matrices, some scattered reads and writes are required.

2. Figures 16 and 19 show that, for quartics, multifrontal solver scales logarithmically, but it is slower than in other cases. The reason for this is that, for p equals four, data fill shared memory in a poor way, and the number of running thread blocks must be higher to process all frontal matrices in separate blocks.
3. One last thing worth noting is that, for quadratics (Figure 14) for 1536 elements, the time needed to generate matrices drops. It happens in every consecutive run of the solver. Our conclusion is that, in this case, data can be divided optimally into blocks without any reminder for more steps, and additionally, multiprocessor cache is used in a better way.

8. Comparison to MUMPS in one dimension

In this section, we present a comparison of our one-dimensional solver with a state-of-the-art MUMPS solver [1, 2, 3] executed on an Intel Core 2 Duo E8500 CPU with 3.16 GHz. The comparison is presented in Figures 20-23 for linear, quadratic, cubic, and quartic B-splines.

From the presented comparison, the implication is that our solver delivers logarithmic execution time, while MUMPS solver delivers quadratic execution time, and our solver is always faster for a large number of elements. The MUMPS solver is faster for a small number of elements for quadratic, cubic, and quartic B-splines (since it is a highly-optimized code). Notice the logarithmic scale on the horizontal axis which implies, for example, the following: for quartic B-splines presented in Figure 23, the MUMPS solver is faster up to 500 elements, but it is slower from 500 up to 2500 elements (where we finished our experiment). For a large number of elements, our solver is two orders of magnitude faster for linear B-splines and one order of magnitude faster for quartic B-splines.

9. Conclusions

We introduced the methodology for concurrent integration and solution of linear systems produced by B-spline-based finite elements delivering higher-order global continuity of the solution. The methodology deliver logarithmic execution time for polynomial orders $p = 1, 2, 3, 4, 5$ that is C^0-linears, C^1-quadratic, C^2-cubic, C^3-quartics and C^4-quintics. The developed model was implemented and tested on a NVIDIA CUDA GPU confirming the logarithmic scalability.

The extension of the model to two-dimensional problems is already described in the paper [16]. Future work will include the extension of the methodology to two- and three-dimensional problems.

Acknowledgements

The work of KK was supported by the Polish National Science Center (grant no. NN 519 447739), and the work of MP was supported by the Polish National Science Center (grants no. 2011/01/D/ST6/02023 and 2012/07/B/ST6/01229).

References

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

[2] Amestoy P. R., Duff I. S., Koster J., L'Excellent J.-Y.: A fully asynchronous multifrontal solver using distributed dynamic scheduling. SIAM *Journal of Matrix Analysis and Applications*, 23, 1, pp. 15–41, 2001

[3] Amestoy P. R., Guermouche A., L'Excellent J.-Y., Pralet S.: Hybrid scheduling for the parallel solution of linear systems. *Parallel Computing*, 32, pp. 136–156, 2005.

[4] Calo V., Gao L., Paszyński M.: Fast algorithms for explicit dynamics, In: *10th World Congress on Computational Mechanics*, WCCM 2012, Sao Paolo, Brasil, 8–13 July, 2012.

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

[6] Demkowicz L.: *Computing with hp-Adaptive Finite Element Method. Vol. I. One and Two Dimensional Elliptic and Maxwell Problems*. Chapmann & Hall / CRC Applied Mathematics & Nonlinear Science, 2006.

[7] Demkowicz L., Kurtz J., Pardo D., Paszyński M., Rachowicz W., Zdunek A.: *Computing with hp-Adaptive Finite Element Method. Vol. II. Frontiers: Three Dimensional Elliptic and Maxwell Problems*. Chapmann & Hall / CRC Applied Mathematics & Nonlinear Science, 2007.

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

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

[10] Fialko S.: A block sparse shared-memory multifrontal finite element solver for problems of structural mechanics. *Computer Assisted Mechanics and Engineering Sciences*, 16, pp. 117–131, 2009.

[11] Fialko S.: The block subtracture multifrontal method for solution of large finite element equation sets. *Technical Transactions*, 1-NP, 8, pp. 175–188, 2009.

[12] Fialko S.: PARFES: A method for solving finite element linear equations on multicore computers. *Advances in Engineering Software*, 40, 12, pp. 1256–1265, 2010.

[13] Geng P., Oden T. J., van de Geijn R. A.: A Parallel Multifrontal Algorithm and Its Implementation. *Computer Methods in Applied Mechanics and Engineering*, 149, pp. 289–301, 2006.

[14] Giraud L., Marocco A., Rioual J.-C: *Iterative versus direct parallel substructuring methods in semiconductor device modeling. Numerical Linear Algebra with Applications*, 12: 1, pp. 33–55, 2005.

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

[16] Kuźnik K., Paszyński M., Calo V.: Graph Grammar-Based Multi-Frontal Parallel Direct Solver for Two-Dimensional Isogeometric Analysis. *Procedia Computer Science*, 9, pp. 1454-1463, 2012.

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

[18] Paszyńska A., Paszyński M., Grabska E.: Graph transformations for modeling *hp*-adaptive Finite Element Method with mixed triangular and rectangular elements. *Lecture Notes in Computer Science*, 5545, pp. 875–884, 2009.

[19] Paszyńska A., Paszyński M., Grabska E.: Graph transformations for modeling *hp*-adaptive Finite Element Method with triangular elements. *Lecture Notes in Computer Science*, 5103, pp. 604–613, 2008.

[20] Paszyński M., Paszyńska A.: Graph transformations for modeling parallel *hp*-adaptive Finite Element Method. *Lecture Notes in Computer Science*, 4967, pp. 1313–1322, 2008.

[21] Paszyński M., Pardo D., Torres-Verdin C., Demkowicz L., Calo V.: A Parallel Direct Solver for Self-Adaptive hp Finite Element Method. *Journal of Parall and Distributed Computing*, 70, pp. 270–281, 2010.

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

[23] Paszyński M., Schaefer R.: Graph grammar driven partial differential eqautions solver. *Concurrency and Computations: Practise and Experience*, 22, 9, pp. 1063–1097, 2010.

[24] Scott J. A.: Parallel Frontal Solvers for Large Sparse Linear Systems. *ACM Transaction on Mathematical Software*, 29, 4, pp. 395-417, 2003.

[25] Smith B. F., Bjørstad P., Gropp W.: *Domain Decomposition, Parallel Multi-Level Methods for Elliptic Partial Differential Equations.* Cambridge University Press, New York, 1st ed. 1996.

[26] Szymczak A., Paszyński M.: Graph grammar based Petri net controlled direct sovler algorithm. *Computer Science*, 11, pp. 65–79, 2010.

Affiliations

Krzysztof Kuźnik
AGH University of Science and Technology, Department of Computer Sciences, Krakow, Poland

Maciej Paszyński
AGH University of Science and Technology, Department of Computer Sciences, Krakow, Poland

Victor Calo
King Abdullah University of Science and Technology, Thuwal, Saudi Arabia

Received: 2.01.2013
Revised: 2.03.2013
Accepted: 2.03.2013