Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 18 (2013) 1574 – 1583

Procedia Computer Science

International Conference on Computational Science, ICCS 2013

Grammar-Based Multi-Frontal Solver for One Dimensional Isogeometric Analysis with Multiple Right-Hand-Sides

Krzysztof Kuźnik[a], Maciej Paszyński[a], Victor Calo[b]

[a]*AGH University of Science and Technology, Krakow, Poland*
[b]*King Abdullah University of Science and Technology, Thuwal, Saudi Arabia*

Abstract

This paper introduces a grammar-based model for developing a multi-thread multi-frontal parallel direct solver for one-dimensional isogeometric finite element method. The model includes the integration of B-splines for construction of the element local matrices and the multi-frontal solver algorithm. The integration and the solver algorithm are partitioned into basic indivisible tasks, namely the grammar productions, that can be executed squentially. The partial order of execution of the basic tasks is analyzed to provide the scheduling for the execution of the concurrent integration and multi-frontal solver algorithm. This graph grammar analysis allows for optimal concurrent execution of all tasks. The model has been implemented and tested on NVIDIA CUDA GPU, delivering logarithmic execution time for linear, quadratic, cubic and higher order B-splines. Thus, the CUDA implementation delivers the optimal performance predicted by our graph grammar analysis. We utilize the solver for multiple right hand sides related to the solution of non-stationary or inverse problems.

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

1. Introduction

The classical higher order finite element method [5, 6] delivers higher order polynomials, but the global continuity is only C^0 between particular mesh elements. The isogeometric finite element method utilizes the B-splines as basis functions, and thus it delivers C^k global continuity [4].

The multi-frontal solver is the state-of-the art algorithm for solving linear systems of equations [8, 12], and it is a generalization of the frontal solver algorithm [14, 7].

The parallelization of the multi-frontal direct solver may target the distributed memory, shared memory or hybrid architectures. The distributed memory parallelization requires either partitioning of the computational domain into sub-domains with overlapping or non-overlapping sub-domains [19, 20], redistribution of the global matrix [13], or redistribution of the elimination tree into processors [16, 17, 18].

Shared memory parallel multi-frontal solvers are designed to store the entire matrix in the shared memory and to perform the matrix operations concurrently. Some examples of the shared memory direct solver are [9, 10, 11]. The matrix is partitioned there into blocks and the operations are performed concurrently over each block. The main disadvantage of this method is that the parallelization is performed based on the analysis of the structure of

Email address: paszynsk@agh.edu.pl (Maciej Paszyński)
URL: http://home.agh.edu.pl/~paszynsk (Maciej Paszyński)

1877-0509 © 2013 The Authors. Published by Elsevier B.V.
Selection and peer review under responsibility of the organizers of the 2013 International Conference on Computational Science
doi:10.1016/j.procs.2013.05.325

the matrix, while we claim that the better performance can be obtained by performing the parallelization based on the analysis of the elimination tree, which is the subject of this paper.

We target our solver into GPU, which is a hybrid architecture. We tested our implementation on NVIDIA Tesla C2070 with 6GB memory and total 448 CUDA cores.

The parallel multi-frontal solver has been implemented for C^0 two dimensional hp-finite element method [16]. For one dimensional regular mesh, the construction of the assembly tree is straightforward, since it is just a binary tree with element matrices at leaf nodes.

In this paper we present the generalization of the grammar based solver for one dimensional finite difference simulation [15]. The graph grammar model allows us to investigate if concurrency is hidden within the algorithm. It is done by analyzing the partial order of execution of the grammar productions, and identification of sets of productions that can be executed concurrently [18].

In this paper we generalize the idea into one dimensional B-spline-based finite element method, delivering C^k global continuity of the solution. The methodology derived here implies logarithmic execution time 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 and higher order B-splines. The performance of this algorithmic implementation is in line with our complexity estimates.

The solver is designed for solution of the problems with multiple right hand sides, where all the right hand sides are not known at the beginning, they are rather computed on fly. The applications of such the solver involve solution of non-stationary problems with constant material data, where the boundary conditions are incorporated at the root node of the elimination tree, enabling for re-utilization of LU factorizations from previous time step. Another application is the solution of an inverse problem, where we have a sequence of several right hand sides. For a detailed description of the model Laplace problem and B-spline based finite element method please refer to [21].

2. Isogeometric multi-frontal solver expressed by graph grammar productions

Graph grammar productions presented in this section represent all the basic tasks that are needed to solve 1D differential equation using the isogeometric finite element method and parallel multifrontal solver for a linear system of equations.

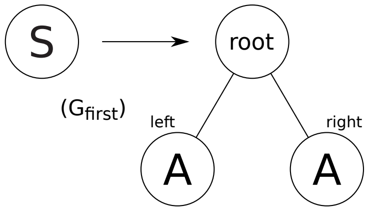

Fig. 1. Initial production for generation of elimination tree

2.1. Generation of elimination tree

Set of (G) productions is responsible for a generation of the elimination tree. The computational domain is partitioned into finite elements by applying:

- (G_{first}) - a starting production which generates *root* of the elimination tree and two A nodes, each containing a direction attribute indicating its position in the domain. Here each of A nodes represents half of the domain. See figure 1
- $(G)^{left}$, (G) and $(G)^{right}$ - intermediate productions splitting each domain part into 2 new parts. Direction attributes propagate during the generation as shown in the figure 2.
- $(G_{last})_p$ - at the leaves level this is the production which produces final domain partitioning. It also has direction attributes. Moreover G_{last} production differs depending on B-spline degree (p) used for computations. As shown in the figure 3 the production $(G_{last})_p$ generates $p + 1$ final nodes. $(G_{last})_1$ is used for linear B-splines, $(G_{last})_2$ for quadratic, etc.

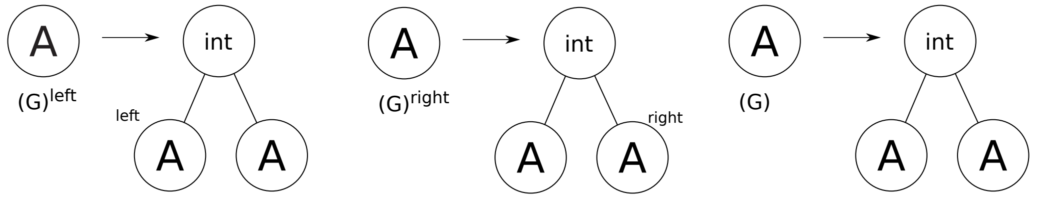

Fig. 2. Intermediate productions for generation of elimination tree

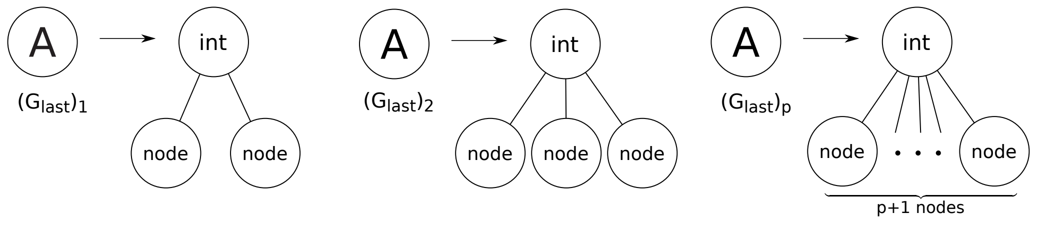

Fig. 3. Final production for generation of elimination tree

2.2. *Generation of local stiffness matrices*

After applying productions (G) the computational domain $\Omega = [0, 1]$ is partitioned into N finite elements $\Omega = \cup_{k=1,...,N}[\xi_k, \xi_{k+1}] = \cup_{k=1,...,N}[\frac{k-1}{N}, \frac{k}{N}]$. Now it is time to assemble local to element stiffness matrices and force vectors. These matrices will be also frontal matrices for multifrontal solver. Each frontal matrix is generated by computing the value of $b\left(N_{j,p}(x), N_{i,p}(x)\right)$ for B-splines defined over the element e_k. Size of the frontal matrix depends on B-spline basis degree:

- For linear B-splines ($p = 1$), there are two basis functions $N_{k,1}$ and $N_{k+1,1}$ with support over an element e_k. This means that element frontal matrix size is $2 \times 2 = 4$, as it is illustrated formula 1 below.

$$\begin{bmatrix} b\left(N_{k,1}(x), N_{k,1}(x)\right) & b\left(N_{k,1}(x), N_{k+1,1}(x)\right) \\ b\left(N_{k+1,1}(x), N_{k,1}(x)\right) & b\left(N_{k+1,1}(x), N_{k+1,1}(x)\right) \end{bmatrix} \tag{1}$$

- 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 an element e_k. The element frontal matrix has $3 \times 3 = 9$ entries, see formula 2 below.

$$\begin{bmatrix} b\left(N_{k,2}(x), N_{k,2}(x)\right) & b\left(N_{k,2}(x), N_{k+1,2}(x)\right) & b\left(N_{k,2}(x), N_{k+2,2}(x)\right) \\ b\left(N_{k+1,2}(x), N_{k,2}(x)\right) & b\left(N_{k+1,2}(x), N_{k+1,2}(x)\right) & b\left(N_{k+1,2}(x), N_{k+2,2}(x)\right) \\ b\left(N_{k+2,2}(x), N_{k,2}(x)\right) & b\left(N_{k+2,2}(x), N_{k+1,2}(x)\right) & b\left(N_{k+2,2}(x), N_{k+2,2}(x)\right) \end{bmatrix} \tag{2}$$

- Generally, for B-splines of degree p there is $p+1$ functions with support over element e_k, denoted $N_{k,p}, ..., N_{k+p,p}$ and the element frontal matrix has $(p + 1)^2$ entries.

Productions $(A_k)_p$ are responsible for generation of local frontal matrices (LFM). They generate stiffness matrix and force vector for element e_k when B-splines of degree p are used (figure 4).

2.3. *Merging of local stiffness matrices*

Local stiffness matrices are also frontal matrices for the multifrontal solver. Productions (M) are responsible for merging the frontal matrices at defined level of the elimination tree. There are two types of this merge:

- $(M_{first})_p$ - a production for the initial merge depending on B-spline degree p. This is merging at leaf level of elimination tree - $p+1$ matrices are merged for B-spline of degree p. Note that at leaf level $p+1$ consecutive frontal matrices have only one common degree of freedom. See example for $(M_{first})_1$ on figure 5.

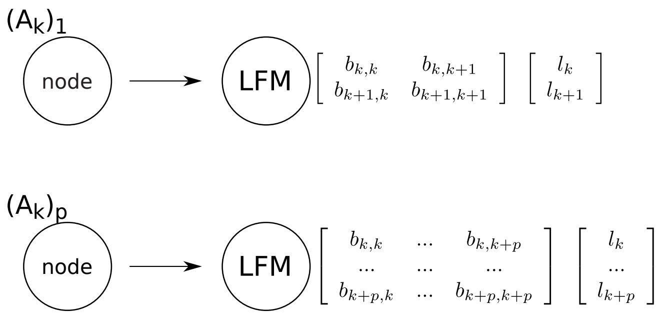

Fig. 4. Productions for generation of LFM

- $(M)_p$ - a production for merging up to the root of elimination tree. This production always merges 2 nodes representing consecutive parts of the domain. However, depending on B-spline degree there is a different number of rows and a different number of common degrees of freedom. For B-spline of degree p there are matrices of size $2p \times 2p$ with $2p$ common degrees of freedom. See figure 6.

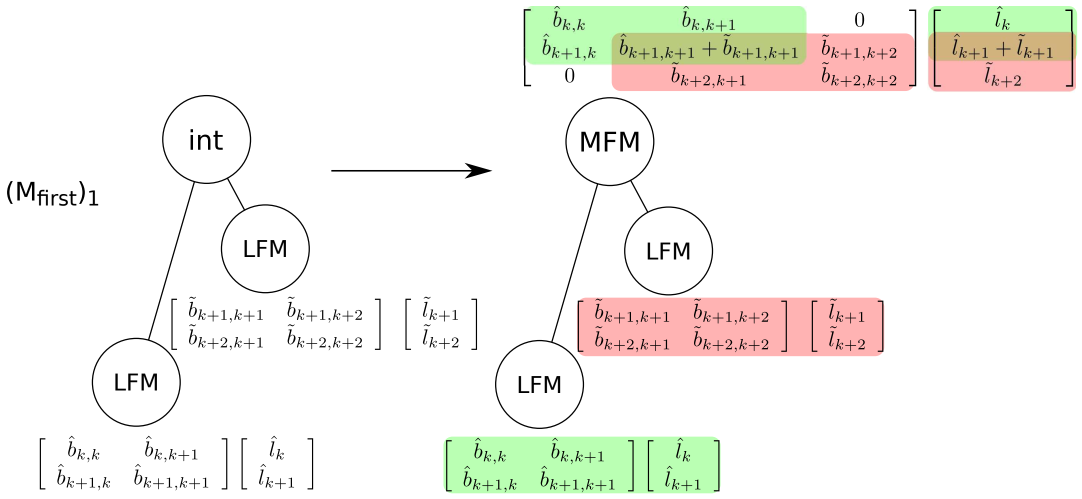

Fig. 5. Production (M_{first}) for merging of **LFM** at leaf level (linear B-splines)

2.4. *Elimination of fully assembled rows*

Productions (E) are responsible for eliminating fully assembled rows from merged frontal matrices (MFM). After eliminating fully assembled row(s) with Gaussian elimination, obtained Schur complement becomes new intermediate local frontal matrix (ILFM) ready for merging at the next level of the elimination tree. There are two different types of this elimination that can be distinguished:

- $(E_{first})_p$ - a production for eliminating exactly one fully assembled row obtained after the initial merge $(M_{first})_p$. Depending on the degree p of a B-spline basis used in FEM the row length is different. It is exactly $2p + 1$. See the example for $(E_{first})_1$ on the figure 7.
- $(E)_p$ - a production for eliminating fully assembled rows at intermediate levels of the elimination tree. For B-spline of degree p it eliminates exactly p rows. Each row length is exactly $3p$. See the example for $(E)_1$ on the figure 8.

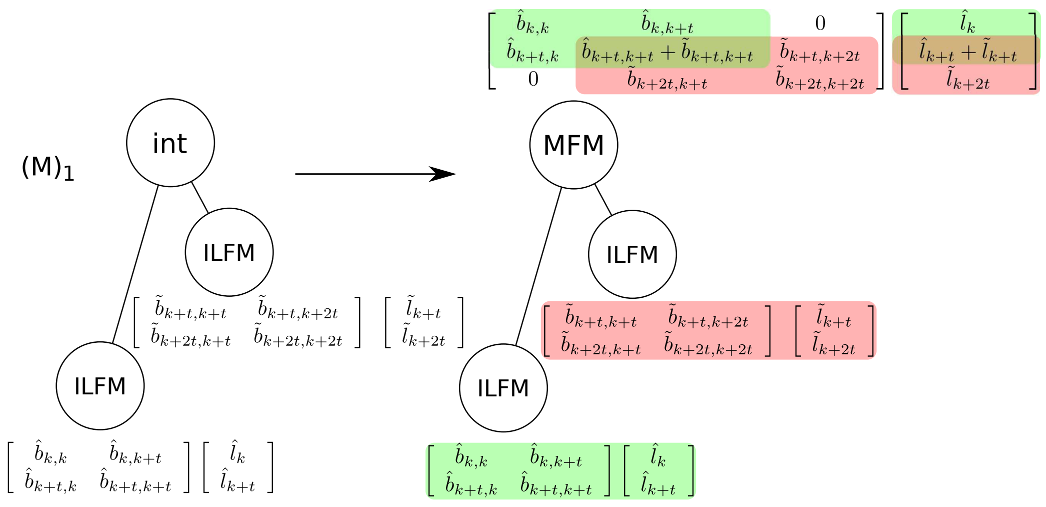

Fig. 6. Production (M) for merging of **ILFM** at intermediate level (linear B-splines)

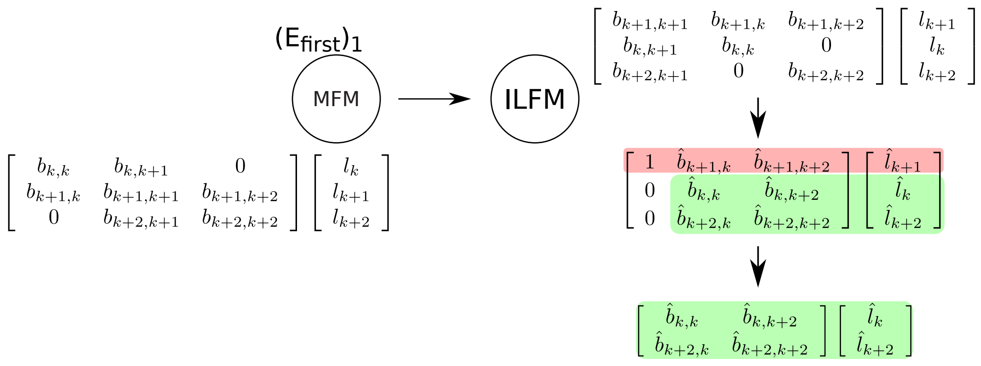

Fig. 7. Production (E_{first}) for eliminating fully assembled row from **MFM** at leaf level (linear B-splines)

2.5. *Root problem*

After last two frontal matrices are merged there is a system of equations where all degrees of freedom are fully assembled. This is called the root problem and a production responsible for this is $(R)_p$. This system is again solved by the Gaussian elimination. Coefficients obtained from the solution of this system are also part of the final solution to the DE. They are respectively u_1, ... , u_p, $u_{N/2+1}$, ... , $u_{N/2+p}$, u_{N+1}, ... ,u_{N+p}. Having them it is possible to obtain the final solution of FEM by recursive backward substitutions. The figure 9 represents an example $(R)_1$ production for linear B-splines.

2.6. *Backward substitution*

Productions (B) are responsible for the recursive backward substitution. They work analogically to building elimination tree by productions (G). At this point each ILFM node contains p fully assembled rows which need $2p$ coefficients to be solved (to obtain new coefficients for those fully assembled degrees of freedom). At the leaf level there are nodes that contain exactly one row which also depends on $2p$ coefficients from the higher level of the elimination tree. Hence again there are two types of this production:

- $(B)_p$ - a production for calculating p coefficients at intermediate levels of the elimination tree. Evaluation is done with the usage of $2p$ coefficients calculated at higher levels of the tree. The figure 10 is an example of this production for linear B-splines.

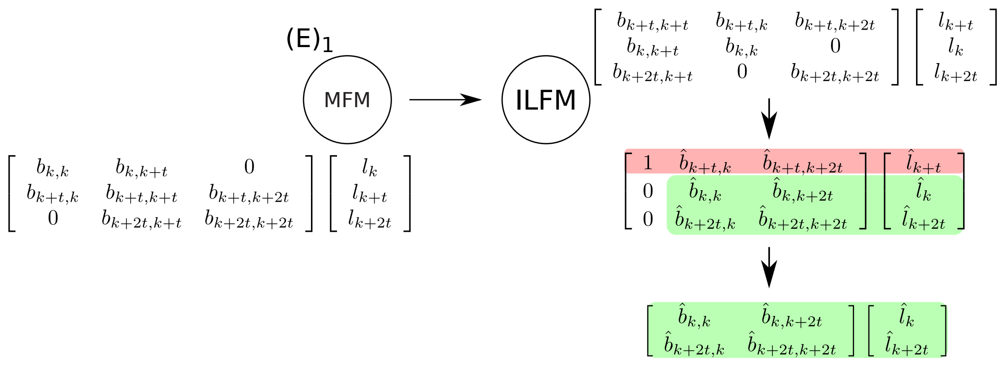

Fig. 8. Production (*E*) for eliminating fully assembled row from **MFM** (linear B-splines)

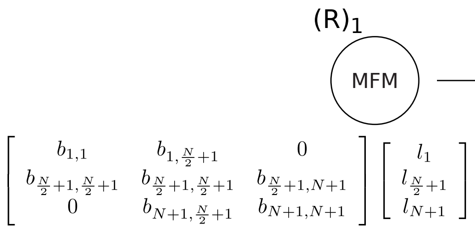

Fig. 9. Production (*R*) for solving the root problem (linear B-splines)

- $(B_{last})_p$ - a production for calculating one coefficient at the leaf level of the elimination tree. The evaluation is done with the usage of $2p$ coefficients calculated at higher level of the tree. Figure 11 is an example of this production for linear B-splines.

2.7. *Scheduling tasks for the solver execution*

When all productions are defined it is possible to schedule all basics tasks needed for the solver execution. At first, the domain is partitioned and the elimination tree is constructed with productions (*G*). When the construction is complete the frontal matrices are constructed at the leaves of the elimination tree with productions (*A*). Then with productions (*M*) and (*E*) applied alternately the frontal matrices are merged and fully assembled degrees of freedom are eliminated. In the end, the root problem is solved with production (*R*). This is followed by recursive backward substitutions (*B*) from the root down to the leaves.

The scheduling for quadratic and higher order B-splines differs from the one discussed for linear B-splines. For quadratic B-splines production (M_{first}) merges three matrices and only one degree of freedom is fully assembled and ready to be eliminated. In the following steps two matrices are merged as with linear B-splines. However, after each merge two degrees of freedom are fully assembled and eliminated. Whole process can be generalized for higher order B-splines.

The crucial property of this algorithm is that productions (*A*), (*M*), (*E*) and (*B*) at each level of the elimination tree process independent sets of data – so they can be applied concurrently.

3. Numerical results

In this section we present a comparison of our one dimensional solver with state of the art MUMPS solver [1, 2, 3] executed on Intel Quad CPU Q9400 with 2.66GHz clock, 8GB of memory. We also present the solver performance for processing multiple right hand sides. The GPU runs have been executed on NVidia Tesla c2070, 6GB memory, 448 CUDA cores, each one with 1.15GHz clock. The comparison is presented in Figure 12 for linear, quadratic, cubic and quartic B-splines. All plots are in linear scale. We believe the non-logarithmic scale emphesizes the huge difference in performance between the GPU and CPU implementations.

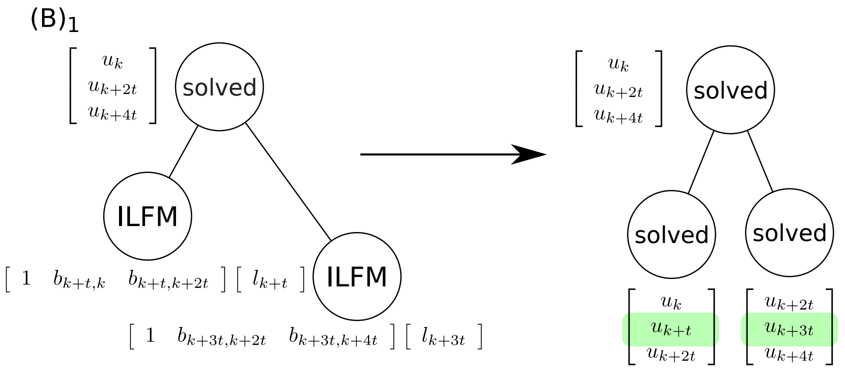
(B)$_1$
$\begin{bmatrix} u_k \\ u_{k+2t} \\ u_{k+4t} \end{bmatrix}$
solved
ILFM
$\begin{bmatrix} 1 & b_{k+t,k} & b_{k+t,k+2t} \end{bmatrix} \begin{bmatrix} l_{k+t} \end{bmatrix}$
ILFM
$\begin{bmatrix} 1 & b_{k+3t,k+2t} & b_{k+3t,k+4t} \end{bmatrix} \begin{bmatrix} l_{k+3t} \end{bmatrix}$
$\begin{bmatrix} u_k \\ u_{k+2t} \\ u_{k+4t} \end{bmatrix}$
solved
solved
solved
$\begin{bmatrix} u_k \\ u_{k+t} \\ u_{k+2t} \end{bmatrix}$
$\begin{bmatrix} u_{k+2t} \\ u_{k+3t} \\ u_{k+4t} \end{bmatrix}$

Fig. 10. Production (*B*) for backward substitution at intermediate levels of elimination tree (linear B-splines)

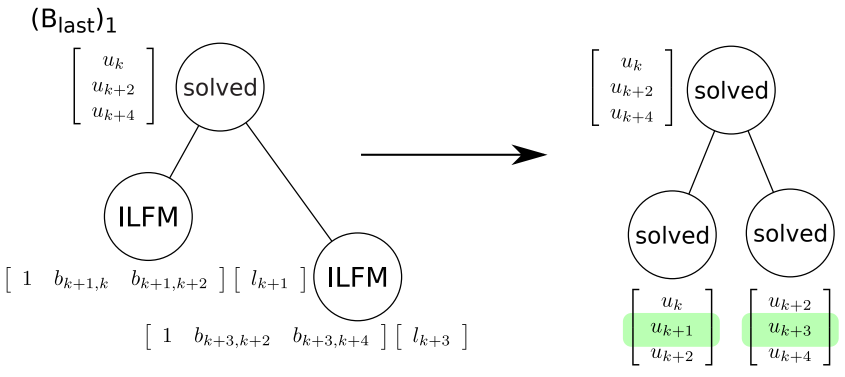
(B$_{last}$)$_1$
$\begin{bmatrix} u_k \\ u_{k+2} \\ u_{k+4} \end{bmatrix}$
solved
ILFM
$\begin{bmatrix} 1 & b_{k+1,k} & b_{k+1,k+2} \end{bmatrix} \begin{bmatrix} l_{k+1} \end{bmatrix}$
ILFM
$\begin{bmatrix} 1 & b_{k+3,k+2} & b_{k+3,k+4} \end{bmatrix} \begin{bmatrix} l_{k+3} \end{bmatrix}$
$\begin{bmatrix} u_k \\ u_{k+2} \\ u_{k+4} \end{bmatrix}$
solved
solved
solved
$\begin{bmatrix} u_k \\ u_{k+1} \\ u_{k+2} \end{bmatrix}$
$\begin{bmatrix} u_{k+2} \\ u_{k+3} \\ u_{k+4} \end{bmatrix}$

Fig. 11. Production (B_{last}) for backward substitution at leaf level of elimination tree (linear B-splines)

In order to understand these results better, we have performed separate measurements for the time spend by GPU and CPU on the integrations, and separately for the factorizations. The comparison is presented in Figures 13 and 14. From the comparison it implies that the 1D GPU algorithm has a logarithmic scalability, while the CPU one delivers linear scalability. The 1D GPU algorithms is up to two orders of magnitude faster than the CPU one. Execution times for GPU confirm the theoretical model presented here with graph grammars. For relatively small problem sizes it is possible to achieve logarithmic growth of time needed to solve the differential equation. "Relatively small" means that we are able to simulate situation where the number of concurrently executed threads is equal to the number of frontal matrices merged at the given level of the elimination tree. Please note that the number of floating point operations in both versions of the solver (CPU and GPU) are of the same order of magnitude as both solvers are multifrontal solvers. Such great performance of the GPU implementation is a result of a special task scheduling and GPU ability to concurrently process huge amount of data.

Solving multiple differential equations (with a common stiffness matrix and different forcing vectors) at the same time is the greatest strength of the implemented solver. Figure 15 presents execution time measured for basis functions of order $p = 1, 2, 3, 4, 5$. The number of elements in each case was equal $128(p + 1)$.

The plot on the figure 15 is interesting because of a few reasons:

- For lower order basis functions ($p = 1, 2, 3$) the cost of calculating 256 RHSes instead of 1 is close to none. For $p = 1$ its hardly tens of **microseconds**.
- Execution time does not grow uniformly – it forms steps, which length is 16. Every 16 RHSes the GPU is required to allocate a new block of threads to process data.
- When the number of RHSes is divisible by 16 the solver runs faster. This means that GPU uses its full potential and does not need to handle idle threads in a special way.

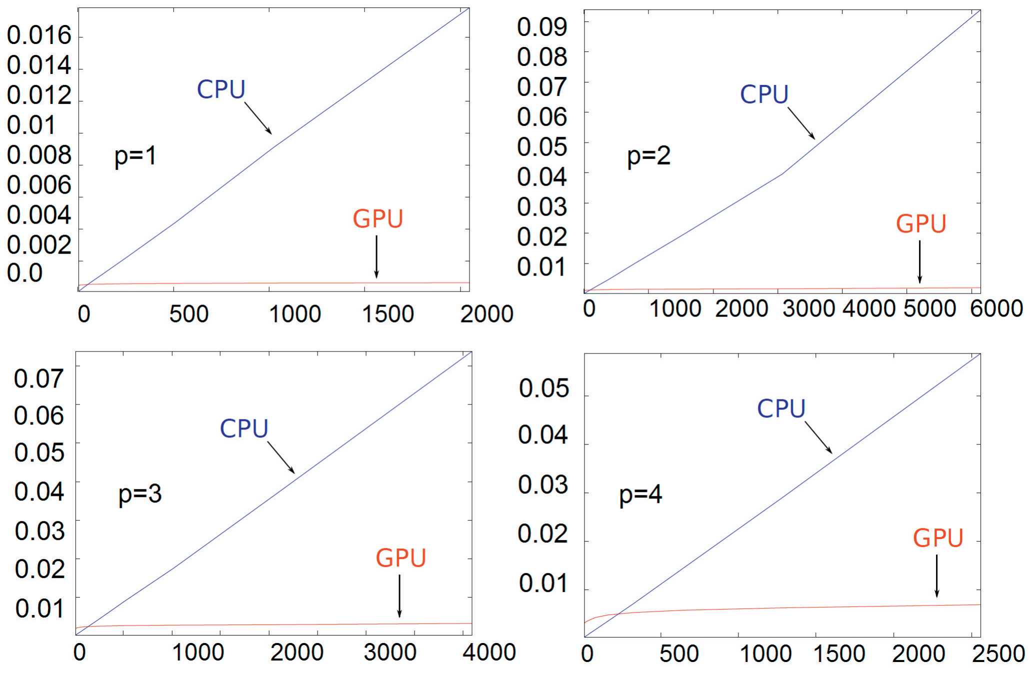

Fig. 12. Comparison of GPU execution time with MUMPS solver CPU execution time for linear, quadratic, cubic and quartic B-splines in one dimension. Horizontal axis denotes number of unknowns, vertical axis denotes time in seconds.

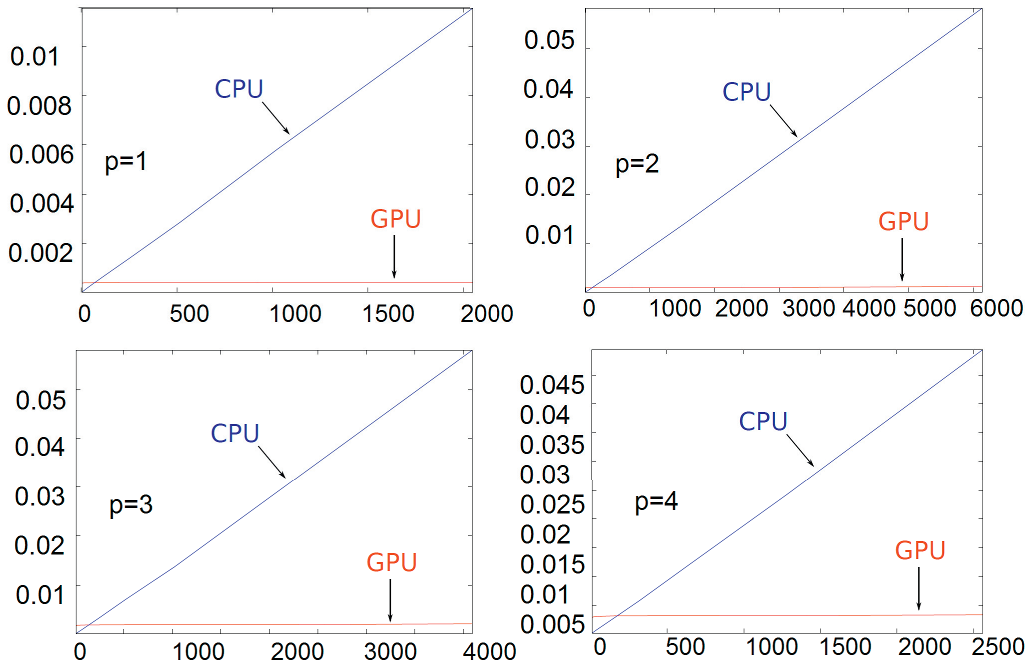

Fig. 13. Comparison of GPU execution time with MUMPS solver CPU execution time for the integration of linear, quadratic, cubic and quartic B-splines in one dimension. Horizontal axis denotes number of unknowns, vertical axis denotes time in seconds.

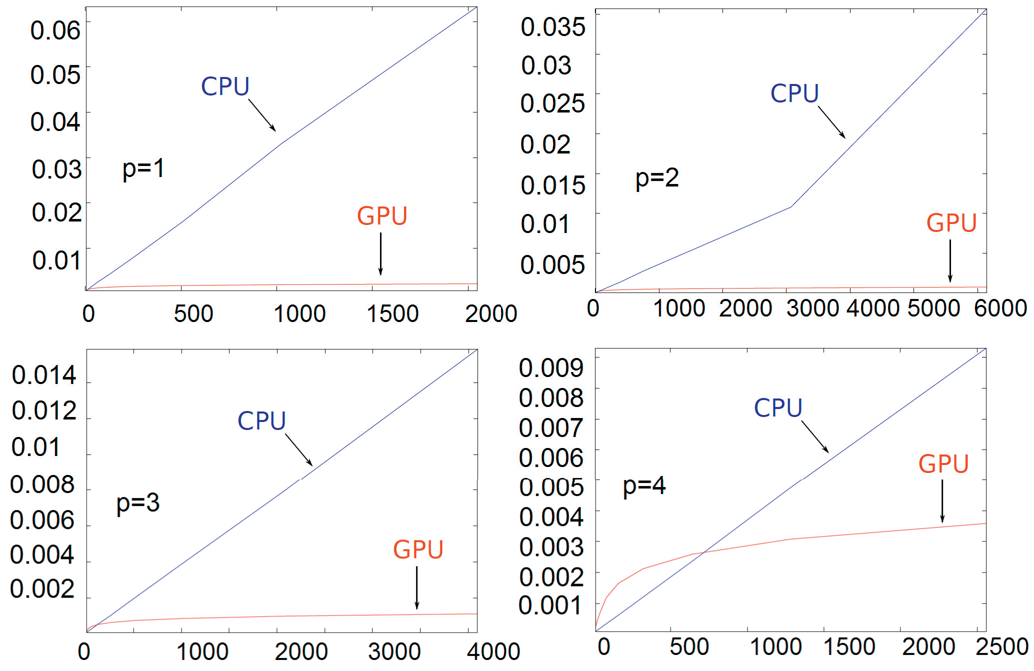

Fig. 14. Comparison of GPU execution time with MUMPS solver CPU execution time for factorization with linear, quadratic, cubic and quartic B-splines in one dimension. Horizontal axis denotes number of unknowns, vertical axis denotes time in seconds.

4. 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, C^1, C^2, C^3 and C^4 respectively. The developed model was implemented and tested on NVIDIA GPU confirming the logarithmic scalability. We also presented the linear scalability of the solver when processing multiple right hand sides.

Acknowledgment

The work reported in this paper was partially supported by the Polish MNiSW grant no. NN 519 447739 and DEC-2011/01/B/ST6/00674.

References

[1] P. R. Amestoy, I. S. Duff and J.-Y. L'Excellent, Multifrontal parallel distributed symmetric and unsymmetric solvers, in Comput. Methods in Appl. Mech. Eng. 184 (2000) 501-520.
[2] P. R. Amestoy, I. S. Duff, J. Koster and J.-Y. L'Excellent, A fully asynchronous multifrontal solver using distributed dynamic scheduling, SIAM Journal of Matrix Analysis and Applications, 23, 1 (2001) 15-41.
[3] P. R. Amestoy and A. Guermouche and J.-Y. L'Excellent and S. Pralet, Hybrid scheduling for the parallel solution of linear systems. Accepted to Parallel Computing (2005).
[4] J. A.Cottrel, T. J. R. Hughes, Y. Bazilevs *Isogeometric Analysis. Toward Integration of CAD and FEA*, Wiley, 2009.
[5] L. Demkowicz, *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.
[6] L. Demkowicz, J. Kurtz, D. Pardo, M. Paszyński, W. Rachowicz, A. Zdunek *Computing with hp-Adaptive Finite Element Method. Vol. II. Frontiers: Three Dimensional Elliptic and Maxwell Problems*. Chapmann & Hall / CRC Applied Mathematics & Nonlinear Science, 2007.
[7] I. S. Duff, J. K. Reid, *The multifrontal solution of indefinite sparse symmetric linear systems*. ACM Transactions on Mathematical Software. 9, 1983; p. 302–325.

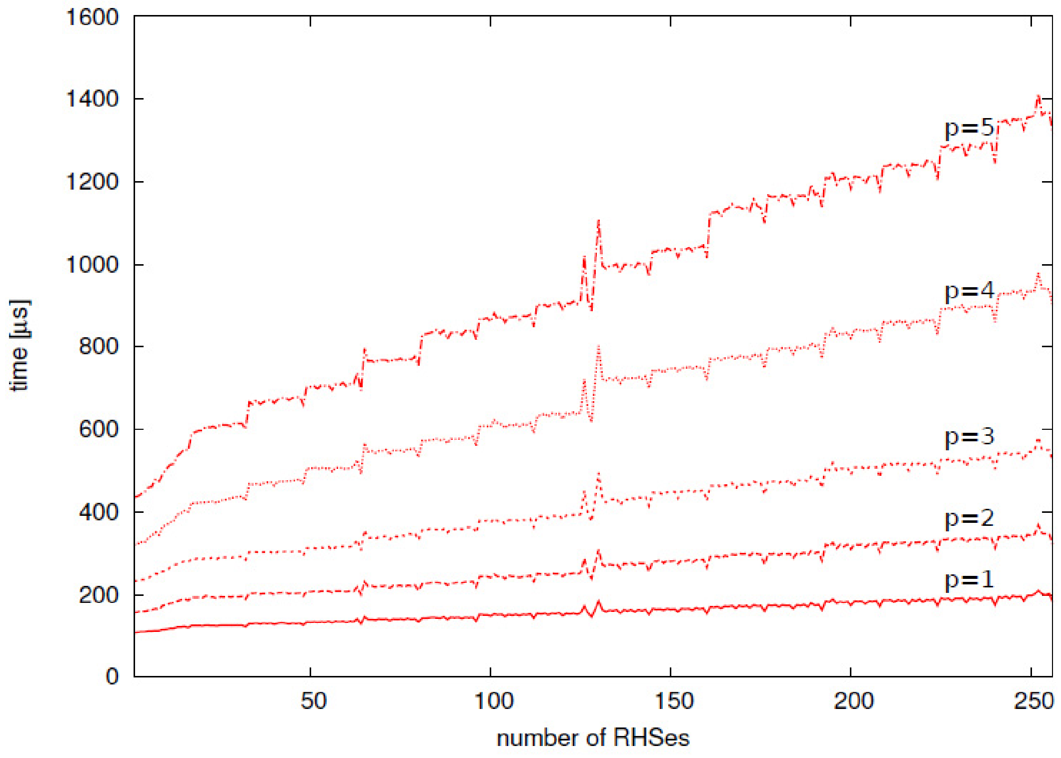

Fig. 15. Time for growing number of RHSes

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

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

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

[11] S. Fialko, *PARFES: A method for solving finite element linear equations on multi-core computers*. Advances in Engineering Software, 40, 12, 2010; p. 1256–1265.

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

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

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

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

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

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

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

[19] J. A. Scott, Parallel Frontal Solvers for Large Sparse Linear Systems, ACM Trans. on Math. Soft., 29, 4 (2003) 395-417.

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

[21] K. Kuźnik, M. Paszyński, V. Calo, *Grammar based multi-frontal solver for isogeometric analysis in 1D* Computer Science, 2013, in press.