Publishing House
AKAPIT

COMPUTER METHODS IN MATERIALS SCIENCE
Informatyka w Technologii Materiałów

Vol. 12, 2012, No. 1

MULTI-FRONTAL MULTI-THREAD DIRECT SOLVER FOR FINITE ELEMENT SIMULATION OF STEP-AND-FLASH IMPRINT LITHOGRAPHY

PAWEŁ OBROK, MACIEJ PASZYŃSKI*

AGH University of Science and Technology, Department of Computer Science,
Al. Mickiewicza 30, 30-059 Kraków, Poland
**Corresponding author: paszynsk@agh.edu.pl*

Abstract

The paper presents the multi-thread multi-frontal direct solver for shared memory architectures. The solver algorithm consists in a sequence of tasks executing recursive forward eliminations and backward substitutions over the constructed elimination tree. The tasks have been grouped into the sets of independent tasks that can be executed in parallel. The computational problem involves two dimensional model of the linear elastivity with thermal expansion coefficient. The finite element method model is used to simulate the Step-and-Flash Imprint Lithography (SFIL), a modern patterning process utilizing photopolymerization to replicate the topography of a template onto a substrate. The multi-thread multi-frontal direct solver has been implemented and tested on NVIDIA CUDA graphic card environment, delivering O($N\log N$) execution time and O($N^{1.5}$) memory usage.

Key words: Multi-frontal direct solver, Finite Element Method, Step-and-Flash Imprint Lithography, NVIDIA CUDA

1. INTRODUCTION

In this paper we present a new multi-frontal solver algorithm for two dimensional finite element method computations dedicated to the multi-core architectures. The multi-frontal solver is the best known algorithm for solving sparse linear systems resulting from finite element simulations. The algorithm performs the LU factorization of the system of linear equations. It is an extension of the frontal solver algorithm, proposed in Irons 1970. The frontal solver browses finite elements, one-by-one, to aggregate degrees of freedom (d.o.f.). Fully assembled d.o.f. are eliminated from the single front matrix. The multi-frontal solver (Duff & Reid, 1983; Geng et al., 2006; Paszyński et al., 2010) generalized the idea into the case of multiple fronts. The solver presented in this paper is the generalization of the multi-thread multi-core solver developed for finite difference method (Obrok et al., 2010). The generalization concerns the application of the solver to the finite element method computations of a linear elasticity problem (Hughes, 2000) utilized to simulate Step-and-Flash Imprint Lithography (SFIL). The solver algorithm utilizes the ideas developed by Paszyński and Schaefer (2010) and Szymczak et al. (2010) for shared memory architectures. The multi-frontal solver is state-of-the art sparse direct solver utilized in industry in sequential implementations such as e.g. BSMFM (Fialko, 2009; Fialko, 2010) as well as parallel implementations such as e.g. PARDISO (Schenk & Gartner, 2002) or MUMPS (Amestoy et al., 2000; Amestoy et al., 2001; Amestoy et al., 2006).

ISSN 1641-8581

2. STEP AND FLASH IMPRINT LITHOGRAPHY

Step and flash imprint lithography (SFIL) is a patterning process utilizing photopolymerization to replicate the topography of a template onto a substrate (Bailey et al., 2002; Colburn et al., 2001, Burns et al., 2004). The SFIL process can be described in the following six steps, as it is illustrated in figure 1.

1) *dispense*. The SFIL process employs a template / substrate alignment scheme to bring a rigid template and substrate into parallelism, trapping the etch barrier in the relief structure of the template,
2) *imprint*. The gap is closed until the force that ensures a thin base layer is reached,
3) *exposure*. The template is then illuminated through the backside to cure etch barrier,
4) *separate*. The template is withdrawn, leaving low-aspect ratio, high resolution features in the etch barrier,
5) *breakthrough etch*. The residual etch barrier (base layer) is etched away with a short halogen plasma etch,
6) *transfer etch*. The pattern is transferred into the transfer layer with an anisotropic oxygen reactive ion etch, creating high-aspect ratio, high resolution features in the organic transfer layer.

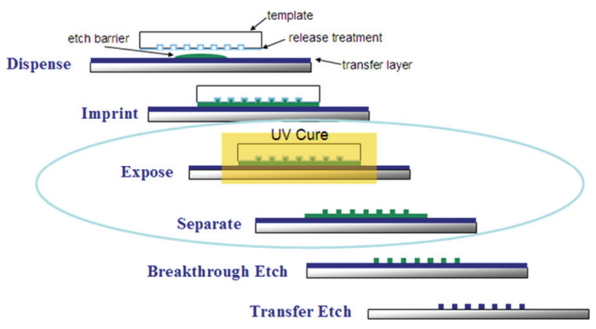


Fig. 1. *Step-and-Flash Imprint Lithography Process.*

The major processing steps of SFIL include: depositing a low viscosity, silicon containing, photocurable etch barrier on to a substrate; bringing the template into contact with the etch barrier; curing the etch barrier solution through UV exposure; releasing the template, while leaving high-resolution features behind; a short, halogen break-through etch; and finally an anisotropic oxygen reactive ion etch to yield high aspect ratio, high resolution features.

Photopolymerization, however, is often accompanied by densification. The interaction potential between photopolymer precursors undergoing free radical polymerization changes from van der Waals' to covalent. The average distance between molecules decreases and causes volumetric contraction. Densification of the SFIL photopolymer (the etch barrier) may affect both the cross sectional shape of the feature and the placement of relief patterns. The exemplary shrinkage of the feature measured after removing the template is presented in figure 2.

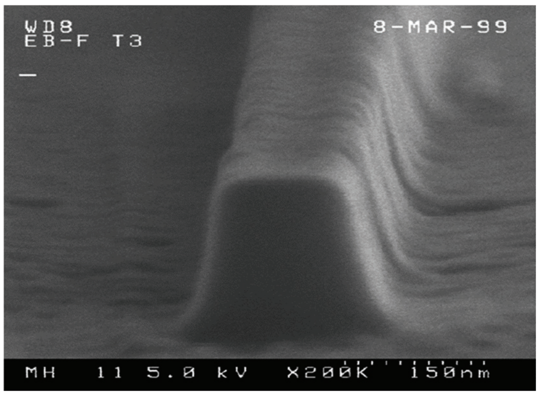


Fig. 2. *Shrinkage of the feature after removal of the template (picture obtained from Prof. Grant C. Wilson from The University of Texas in Austin).*

3. LINEAR ELASTICITY WITH THERMAL EXPANSION COEFFICIENT

The linear elasticity model with thermal expansion coefficient (Hughes, 2000) is used to verify the material response of polymerized networks in cured etch-barrier layers that are formed during the third step of the SFIL process.

Strong formulation

Given $g_i : \Gamma_{D_i} \ni x \to g_i(x) = 0 \in R$, θ, α_{kl} and σ^0_{ij}, find $u_i : \overline{\Omega} \to R$ the displacement vector field such that

$$\sigma_{ij,j} = 0 \text{ in } \Omega \tag{1}$$

$$u_i = g_i \text{ in } \Gamma_{D_i} \tag{2}$$

where σ_{ij} is the stress tensor, defined in terms of the generalized Hooke's law

$$\sigma_{ij} = c_{ijkl}\left(\varepsilon_{kl} + \theta\,\alpha_{kl}\right) + \sigma^0_{ij} \tag{3}$$

c_{ijkl} elastic coefficients (known for a given material), σ^0_{ij} initial stress. ε^0_{ij} initial strain, θ temperature, α_{kl} thermal expansion coefficients, ε_{ij} is the strain tensor, defined to be $u_{(i,j)}$, the symmetric part of the displacement gradients

$$\varepsilon_{ij} = u_{(i,j)} = \frac{u_{i,j} + u_{j,i}}{2} \qquad (4)$$

where u_i displacement vector, $u_{i,j}$ displacement gradients.

Weak formulation

The weak formulation is obtained by multiplying (1) by test functions $w_i \in V_i$ and integrating by parts over Ω.

$$-\int_{\Omega} w_{i,j}\sigma_{ij}\, d\Omega + \int_{\Gamma} w_i\sigma_{ij}n_j\, d\Omega = 0 \qquad (5)$$

Since σ_{ij} is symmetric tensor, then $w_{i,j}\sigma_{ij} = w_{(i,j)}\sigma_{ij}$ (compare Hughes, 2000), and $w_i = 0$ on Γ, we get

$$\int_{\Omega} w_{(i,j)}\sigma_{ij}\, d\Omega = 0 \qquad (6)$$

Finally, we substitute (3) into (6) and get

$$\int_{\Omega} w_{(i,j)}c_{ijkl}u_{(k,l)}\, d\Omega = -\theta\int_{\Omega} w_{(i,j)}c_{ijkl}\alpha_{kl}\, d\Omega - \int_{\Omega} w_{(i,j)}\sigma^0_{ij}\, d\Omega \qquad (7)$$

since $\varepsilon_{ij} = u_{(i,j)}$.

Abstract index-free notation

For implementation issues, the most convenient is the following form:
Find $\mathbf{u} \in \mathbf{V}$ such that

$$\mathbf{a}(\mathbf{w},\mathbf{u}) = -\mathbf{A}(\mathbf{w}) - \mathbf{\Sigma}(\mathbf{w}) \text{ for all } \mathbf{w} \in \mathbf{V} \qquad (8)$$

where:

$$\mathbf{a}(\mathbf{w},\mathbf{u}) = \int_{\Omega} \boldsymbol{\varepsilon}(\mathbf{w})^{\mathbf{T}}\mathbf{D}\boldsymbol{\varepsilon}(\mathbf{u})d\Omega \qquad (9)$$

$$\mathbf{A}(\mathbf{w}) = \theta\int_{\Omega} \boldsymbol{\varepsilon}(\mathbf{w})^{\mathbf{T}}\mathbf{D}\boldsymbol{\alpha}\, d\Omega \qquad (10)$$

$$\mathbf{\Sigma}(\mathbf{w}) = \int_{\Omega} \boldsymbol{\varepsilon}(\mathbf{w})^{\mathbf{T}}\boldsymbol{\sigma}^{\mathbf{0}}\, d\Omega \qquad (11)$$

Here

$$\boldsymbol{\varepsilon}(\mathbf{u}) = \begin{Bmatrix} u_{1,1} \\ u_{2,2} \\ u_{1,2}+u_{2,1} \end{Bmatrix}, \quad \boldsymbol{\varepsilon}(\mathbf{w}) = \begin{Bmatrix} w_{1,1} \\ w_{2,2} \\ w_{1,2}+w_{2,1} \end{Bmatrix} \qquad (12)$$

$$\boldsymbol{\alpha} = \begin{Bmatrix} \alpha_{11} \\ \alpha_{22} \\ 2\alpha_{12} \end{Bmatrix} \qquad (13)$$

where we have assumed the symmetry of $\alpha_{12} = \alpha_{21}$, thus $\alpha_{12} + \alpha_{21} = 2\alpha_{12}$,

$$\boldsymbol{\sigma}^{\mathbf{0}} = \begin{Bmatrix} \sigma^0_{11} \\ \sigma^0_{22} \\ \sigma^0_{12} \end{Bmatrix} \qquad (14)$$

and

$$\mathbf{D} = \begin{Bmatrix} D_{11} & D_{12} & D_{13} \\ D_{12} & D_{22} & D_{23} \\ D_{13} & D_{23} & D_{33} \end{Bmatrix} \qquad (15)$$

and the coefficients of $\mathbf{D}_{IJ}$ are related to coefficients of c_{ijkl} as ilustrated in table 1.

Table 1. Coefficients of $\mathbf{D}_{IJ} = c_{ijkl}$ *tensor*

I / J	i / k	j / l
1	1	1
3	1	2
3	2	1
2	2	2

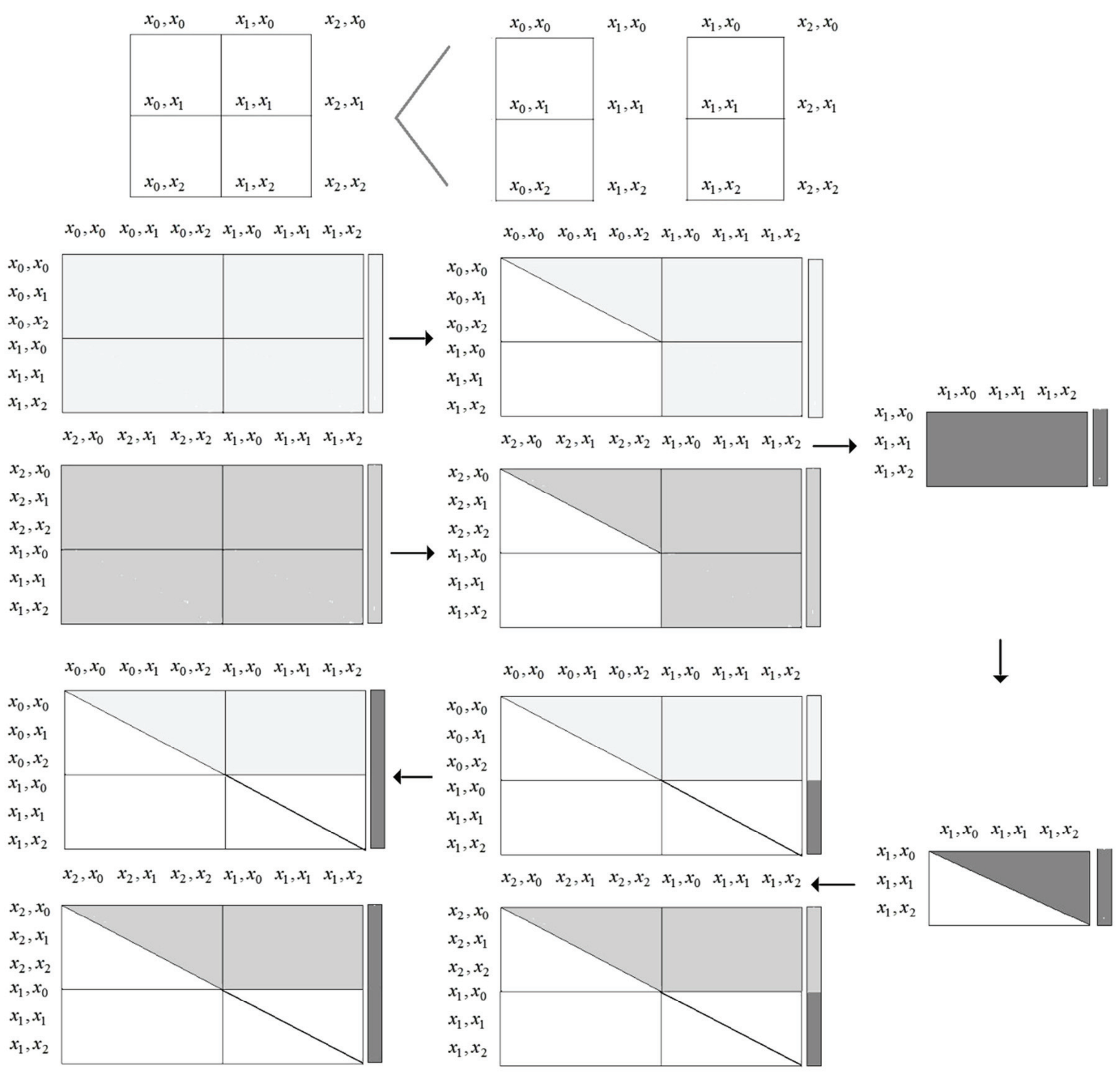
4. MULTI-FRONTAL MULTI-TASKS DIRECT SOLVER

The multi-frontal multi-tasks direct solver starts with creation of the elimination tree, presented on the top panel in figure 3. The exemplary four finite element computational domain is partitioned into vertical slices subdomains. The unknowns called *the degrees of freedom (d.o.f.)* – the horizontal and vertical components of the displacement vector field – are related with finite element vertices. The two subdomains local matrices are created.

In the presented example there are two frontal matrices, the first one related to the first slice of the mesh, the second one related to the second slice of the mesh. Both frontal matrices are constructed in concurrent. Actually we distinguish the creation of particular rows of the frontal matrices. We can name the tasks responsible for constructing of the frontal matrices as

$$\{(AR_1)_k, (AR_2)_k, (AR_3)_k, (AR_4)_k, (AR_5)_k, (AR_6)_k, (AR_7)_k, (AR_8)_k\} \qquad (16)$$

since there are 8 rows in each frontal matrix. Notice that d.o.f. related to external vertices are fully assembled, and the d.o.f. related to the common edge are not fully assembled yet. In other words, the rows of the frontal matrices associated with subdomain external vertices are fully summed up, and these



x0,x0
x1,x0
x2,x0
x0,x1
x1,x1
x2,x1
x0,x2
x1,x2
x2,x2
x0,x0 x0,x1 x0,x2 x1,x0 x1,x1 x1,x2
x2,x0 x2,x1 x2,x2 x1,x0 x1,x1 x1,x2
x1,x0 x1,x1 x1,x2

Fig. 3. *Elimination tree and the summary of the solver algorithm.*

rows can be eliminated by subtracting from the remaining rows. On the other hand, the rows of the frontal matrices associated with subdomain common edge are not fully summed up, only half of the row contribution is assembled into a single frontal matrix. These rows cannot be eliminated yet, but we can subtract the fully assembled rows from them.This procedure is presented in figure 3. In the first frontal matrix we subtract rows associated to external vertices (0,0), (0,1) and (0,2) from the rows associated to the common edge (1,0), (1,1) and (1,2). In the second frontal matrix we subtract rows associated to the external vertices (2,0), (2,1) and (2,2) from the rows associated to the common edge (1,0), (1,1) and (1,2). Notice that these subtractions can be performed in concurrent. We denote the tasks of rows subtraction by

$$\{(ER_{2\text{-}1})_k, (ER_{3\text{-}1})_k, (ER_{4\text{-}1})_k, (ER_{5\text{-}1})_k, (ER_{6\text{-}1})_k, (ER_{3\text{-}2})_k, (ER_{4\text{-}2})_k, (ER_{5\text{-}2})_k, (ER_{6\text{-}2})_k, (ER_{4\text{-}3})_k, (ER_{5\text{-}3})_k, (ER_{6\text{-}3})_k\} \quad (17)$$

where $k = 1,2$ denote the frontal matrix index.

In the next step of the algorithm, the right bottom parts of both frontal matrices are merged together. These parts resulting from the partial forward eliminations correspond to so-called Schur complements. The parts are merged together into a single interface frontal matrix associated with the common edge vertices (1,0), (1,1) and (1,2). All these rows are fully assembled now, and we can perform the full elimination of these rows. In other words we execute the following row subtracting tasks

$$\{(ER_{2\text{-}1})_3, (ER_{3\text{-}1})_3, (ER_{3\text{-}2})_3\} \quad (18)$$

Finally, the solution to the common edge problem is obtained by performing backward substitution over the interface matrix. The solution is also utilized to perform the backward substitutions over the two frontal matrices. The solution from the common edge is copied to the bottom part of the right hand side of both systems, the right bottom parts of both frontal matrices are set to the identity matrix, and backward substitutions are executed. The backward substitutions can be expressed by the following tasks

$$\{(BS_3)_3, (BS_2)_3, (BS_1)_3, (BS_6)_k, (BS_5)_k, (BS_4)_k, (BS_3)_k, (BS_2)_k, (BS_1)_k\} \quad (19)$$

Having all the operations expressed as basic undividable tasks, we can find the dependency relation between tasks, so the tasks can be grouped into sets of independent tasks that can be executed fully in parallel. The two tasks are dependent if the second one must be executed after the first one. The dependency relation between the aggregation and elimination tasks is the following:

$$\{(AR_i)_k \; D \; (ER_{m-n})_k\} \quad (20)$$

where $k = 1,2$, $I = 1,...,8$ and $m = 2,...,6$ and $n = 1,2,3$.

The dependency relation between the elimination tasks is the following:

$(ER_{2-1})_k$ D $(ER_{3-2})_k$, $(ER_{3-1})_k$ D $(ER_{3-2})_k$, $(ER_{2-1})_k$ D $(ER_{4-2})_k$, $(ER_{4-1})_k$ D $(ER_{4-2})_k$, $(ER_{2-1})_k$ D $(ER_{5-2})_k$, $(ER_{5-1})_k$ D $(ER_{5-2})_k$, $(ER_{2-1})_k$ D $(ER_{6-2})_k$, $(ER_{6-1})_k$ D $(ER_{6-2})_k$, $(ER_{3-2})_k$ D $(ER_{4-3})_k$, $(ER_{4-2})_k$ D $(ER_{4-3})_k$, $(ER_{3-2})_k$ D $(ER_{5-3})_k$, $(ER_{5-2})_k$ D $(ER_{5-3})_k$, $(ER_{3-2})_k$ D $(ER_{6-3})_k$, $(ER_{6-2})_k$ D $(ER_{6-3})_k$, $(ER_{4-3})_k$ D (A), $(ER_{5-3})_k$ D (A), $(ER_{6-3})_k$ D (A), (A) D $(ER_{2-1})_3$, (A) D $(ER_{3-1})_3$, $(ER_{2-1})_3$ D $(ER_{3-2})_3$, $(ER_{3-1})_3$ D $(ER_{3-2})_3$ (21)

And the dependency relation between the backward substitution tasks is the following:

$(ER_{3-2})_3$ D $(BS_3)_3$, $(BS_3)_3$ D $(BS_2)_3$, $(BS_2)_3$ D $(BS_1)_3$, $(BS_1)_3$ D $(BS_6)_k$, $(BS_6)_k$ D $(BS_5)_k$, $(BS_5)_k$ D $(BS_4)_k$, $(BS_4)_k$ D $(BS_3)_k$, $(BS_3)_k$ D $(BS_2)_k$, $(BS_2)_k$ D $(BS_1)_k$ (22)

Following Diekert and Rozenberg (1995), using the dependency relation we can automatically schedule tasks into the sets of independent tasks that can be executed in parallel:

$[(AR_1)_1(AR_2)_1(AR_3)_1(AR_4)_1(AR_5)_1(AR_6)_1(AR_7)_1(AR_8)_1(AR_1)_2(AR_2)_2(AR_3)_2(AR_4)_2(AR_5)_2(AR_6)_2(AR_7)_1(AR_8)_2]$
$[(ER_{2-1})_1(ER_{3-1})_1(ER_{4-1})_1(ER_{5-1})_1(ER_{6-1})_1(ER_{2-1})_2(ER_{3-1})_2(ER_{4-1})_2(ER_{5-1})_2(ER_{6-1})_2]$
$[(ER_{3-2})_1(ER_{4-2})_1(ER_{5-2})_1(ER_{6-2})_1(ER_{3-2})_2(ER_{4-2})_2(ER_{5-2})_2(ER_{6-2})_2]$
$[(ER_{4-3})_1(ER_{5-3})_1(ER_{6-3})_1(ER_{4-3})_2(ER_{5-3})_2(ER_{6-3})_2]$
$[(A)]$ $[(ER_{2-1})_3(ER_{3-1})_3]$ $[(ER_{3-2})_3]$
$[(BS_3)_3]$ $[(BS_2)_3]$ $[(BS_1)_3]$ $[(BS_6)_1 (BS_6)_2]$ $[(BS_5)_1 (BS_5)_2]$ $[(BS_4)_1 (BS_4)_2]$
$[(BS_3)_1 (BS_3)_2]$ $[(BS_2)_1 (BS_2)_2]$ $[(BS_1)_1 (BS_1)_2]$ (23)

The groups are denoted by square brackets and are called the Foata Normal Form (Diekert & Rozenberg, 1995). The procedure can be recursively generalized into square shape subdomains with more finite elements. In this case, the subdomain frontal matrices will be associated with vertical slices, the fully assembled d.o.f. that can be eliminated are located inside the slices, and the interface problem will be solved recursively by merging partially assembled rows and eliminating currently fully assembled rows.

Having the solver algorithm expressed as sequence of sets of independent tasks, the algorithm can be mapped into shared memory architecture. The general idea is the following. All tasks from a set of tasks can be executed in concurrent. In other words, we can execute all tasks from the first set at the same time, followed by the global barrier, and then we can execute all tasks from the second set at the same time, and so on. Tasks work on global memory, so no communication between tasks is necessary. The synchronization is restriced to global barriers executed after each set of tasks. NVIDIA CUDA architecture consists in several computing nodes, with multiple cores. In order to be able to execute many tasks in concurrent over the same node, the tasks must be identical but they may work on different input data. In our Foata Normal Form each set of tasks actually consists of the identical tasks (either agregations or subtractions of rows or backward substitution) so this condition is fullfilled. The tasks work on frontal matrices stored in GPU global memory, in a linear fashion – all levels of the elimination tree from leaf nodes up to the root are constructed in the global memory dynamically level by level.

5. NUMERICAL RESULTS

We conclude the paper with numerical results concerning the computations of the linear elasticity with thermal expansion coefficient for the Step-and-

Flash Imprint Lithography, performed by the multi-thread multi-frontal direct solver.

The graphics for the numerical results presented in figure 4 have been obtained by using fortran hp2d code (Demkowicz, 2007), and the scalability of the solver over the GPU shared memory architecture has been presented in figure 5. The time measured corresponds to the total execution time of the solver for given number of degrees of freedom.

The solver has been executed on PC with Intel Core 2 Duo E8400 processor and Nvidia GeForce 275 GTX graphics card with 240 processing cores and with 896 MB of memory.

The solver scheduled the tasks according to the Foata Normal Form presented in section *Multi-thread mulit-frontal direct solver algorithm.*

The computational cost of the two dimensional solver can be estimated in the following way. The solver constructs the binary elimination tree with depth log N. The leaves in the elimination tree contains the systems with $3N^{0.5}$ d.o.f. and we eliminate $N^{0.5}$ d.o.f. by performing concurrent subtractions of rows. The computational cost of this operation is $3N^{0.5}N^{0.5} = O(N)$. Since the depth of the elimination tree is log$N$ the total computational cost is O(Nlog$N$). Summing up the two dimensional solver delivers O(Nlog$N$) execution time.

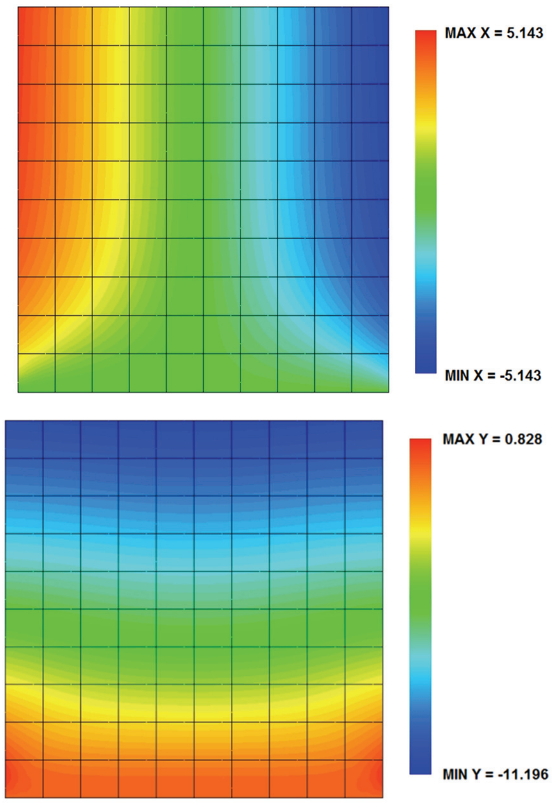


Fig. 4. *The horizontal and vertical components of the resulting displacement vector field.*

The limitation of the solver is the capacity of the graphic card memory. This is because the entire elimination tree with all generated frontal matrices must be kept in GPU memory until the backward substitution stage. Thus the memory usage of the solver is equal to the amount of memory required to store the entire binary elimination tree, with number of nodes equal to the number of elements. The nodes of the elimination tree contain the frontal matrices with O($N^{0.5}$) degrees of freedom. The size of each matrix is $O((N^{0.5})^2) = O(N)$. The number of leaves in elimination tree is O($N^{0.5}$). The tree is binary, thus the total number of nodes is $2N^{0.5} = O(N^{0.5})$. Thus the memory usage of the solver is O($N^{1.5}$). The maximum problem that could be solved on the presented graphic card with 896 MB of memory was $N = 2^{16} =$ 65536 degrees of freedom.

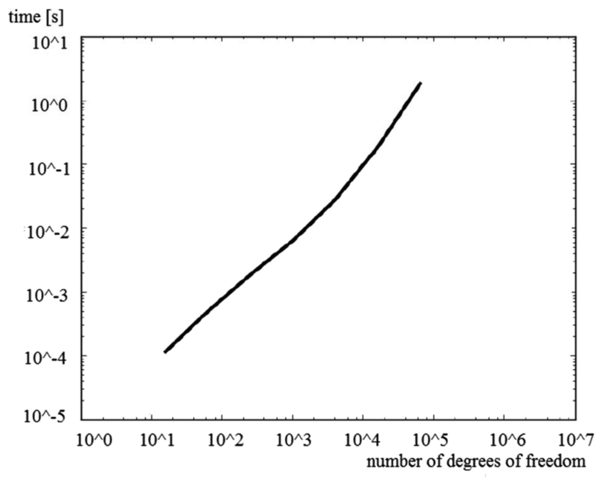


Fig. 5. *The scalability of the solver algorithm.*

The computing time of the proposed solver is compared against the state of the art MUMPS solver executed on CPU. The MUMPS solver results are summarized in table 2. The comparison implies that for 10000 degrees of freedom the GPU solver delivers solution within 0.1 second, while the MUMPS solver delivers solution within 0.2 second. For 65536 degrees of freedom the GPU solver delivers solution within 2 seconds, which is of the same order as the MUMPS solver solution time.

Table 2. Computing time for the 2D MUMPS solver.

Grid size in one dimension	Number of degrees of freedom	Time [s]
100	10000	0.220
200	40000	0.904
300	90000	2.048
400	160000	4.060
500	250000	6.592

The execution times are of the same order, however our solver is parallel, and we compare against sequential MUMPS. The reasons while our parallel solver is not faster than sequential MUMPS solver are the following. First, the MUMPS solver is highly optimized and uses optimized BLAS libraries, such as LAPACK and BLAS while our solver uses home made matrix operation routines (no numerical algebra libraries for GPU are used). Second, the CPU utilized for MUMPS solver is Intel Core 2 Duo E8400 with 3GHz clock, while processing cores on GPU have 650 MHz clock, so the GPU processing cores are 5 times slower than CPU, and CPU is dual core.

6. CONCLUSIONS

In this paper the multi-thread multi-frontal direct solver for shared memory architecture was presented. The solver was dedicated to the two dimensional computations of the linear elasticity with thermal expansion coefficient by means of the finite element method. The solver was expressed as a sequence of tasks involving row subtractions for recursive forward elimination pattern, followed by recursive backward substitutions. The tasks were scheduled according to the obtained Foata Normal Form, representing sets of independent tasks that can be executed in concurrent. The solver delivers $O(N\log N)$ execution time and $O(N^{1.5})$ memory usage.

ACKNOWLEDGEMENTS

The work reported in this paper has been partially supported by MNiSW grant no. 501 120836

REFERENCES

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

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

Amestoy, P. R., Guermouche, A., L'Excellent, J.-Y., Pralet, S., 2006, Hybrid scheduling for the parallel solution of linear systems, *Parallel Computing*, 32, 2, 136-156

Bailey, T. C., Colburn, M. E., Choi, B. J., Grot, A., Ekerdt, J. G., Sreenivasan, S. V., Willson, C. G., 2002, *Step and Flash Imprint Lithography: A Low-Pressure, Room Temperature Nanoimprint Patterning Process. Alternative Lithography. Unleashing the Potentials of Nanotechnology*, C. Sotomayor Torres, Editor, Elsevier, Amsterdam.

Burns, R. L., Johnson, S. C., Schmid, G. M., Kim, E. K., Dickey, D. M. D., Meiring, J., Burns, S. D., Stacey, N. A., Willson, C. G., 2004, Mesoscale modeling for SFIL simulating polymerization kinetics and densification, *Proceeding of SPIE*, 5374, 348-360.

Colburn, M. E., Suez, I., Choi, B. J., Meissi, M., Bailey, T., Sreenivasan, S. V., Ekerdt, J. E., Willson, C. G., 2001, Characterization and modeling of volumetric and mechanical properties for SFIL photopolymers, *Journal of Vacuum Science and Technology* B, 19-6, 2685-2689.

Demkowicz, L. 2007, *Computing with hp adaptive finite element method. Part I. One dimensional elliptic and Maxwell problems,* Chapmann & Hall / CRC Press, Taylor & Francis Group, Boca Raton, London, New York.

Diekert, V., Rozenberg, G., 1995, *The book of traces*, World Scientific, Singapore.

Duff, I. S., Reid, J. K., 1983, The multifrontal solution of indefinite sparse symmetric linear systems, *ACM Transactions on Mathematical Software*, 9, 302-325.

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

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

Fialko, S., 2010, PARFES: A method for solving finite element linear equations on multi-core computers, *Advances in Engineering Software*, 40 (12), 1256-1265.

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

Hughes, T. J. R. 2000, *The Finite Element Method, Linear Statics and Dynamics Finite Element Method Analysis,* Prentice-Hall, New-York.

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

MUMPS: A MUltifrontal Massively Parallel sparse direct Solver, http://www.enseeiht.fr/lima/apo/MUMPS

Obrok, P., Pierzchala, P., Szymczak, A., Paszynski, M., 2010, The grammar-based multi-thread multi-frontal parallel solver with trace theory-based scheduler, *Proceedia Computer Science*, 1, 1987-1995.

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

Paszyński, M., Schaefer, R., 2010, Graph grammar-driven parallel partial differential equation solver, *Concurrency and Computation, Practise and Experience*, 22, 9, 1063-1097.

Schenk, O., Gartner, K., 2002, Two-level dynamic scheduling in PARDISO: Improved scalability on shared memory multiprocessing systems, *Parallel Computing*, 28, 187–197.

Szymczak, A., Paszyński, M., Pardo, D., 2010, Graph Grammar Based Petri Net Controlled Direct Solver Algorithm, *Journal Computer Science*, 11, 65-79.

WIELOWĄTKOWY SOLVER WIELO-FRONTALNY WYKONUJĄCY SYMULACJE METODĄ ELEMENTÓW SKOŃCZONYCH PROCESU NANOLITOGRAFII PRZEZ NAŚWIETLANIE I WYCISKANIE

Streszczenie

W artykule zaprezentowano wielowątkowy solver wielo-frontalny dla architektur o pamięci współdzielonej. Algorytm solvera przedstawiony został w postaci sekwencji tasków wykonujących rekurencyjnie częściowe eliminację oraz podstawiania wstecz w oparciu o skonstruowane drzewo eliminacji. Taski pogrupowane zostały w grupy niezależnych tasków wykonywanych współbieżnie. Algorytm solvera przetestowany został na dwuwymiarowym problemie liniowej sprężystości ze współczynnnikiem rozszerzalności cieplnej. Przeprowadzono obliczenia za pomocą metody elementów skończonych procesu nanolitografii poprzez naświetalnie i wyciskanie, stanowiącej nowoczesną technologię produkcji układów scalonych wykorzystującą zjawisko fotopolimeryzacji. Algorytm solvera został zaimplementowany i przetestowany w środowisku karty graficznej NVIDIA CUDA, osiągając czas wykonania rzędu O($N\log N$) oraz zużycie pamięci rzędu O($N^{1.5}$).

Submitted: June 11, 2011
Submitted in a revised form: September 02, 2011
Accepted: February 21, 2012