GPU-Accelerated 3D Mesh Deformation for Optimization Based on the Finite Element Method

Adam LAMECKI [1], *Adam DZIEKONSKI* [1], *Lukasz BALEWSKI* [1,2], *Grzegorz FOTYGA* [1], *Michal MROZOWSKI* [1]

[1] Faculty of Electronics, Telecommunications and Informatics, Gdańsk University of Technology, Narutowicza 11/12, 80-233 Gdańsk, Poland

[2] EM Invent, Trzy Lipy 3, 80-172 Gdańsk, Poland

adlam@eti.pg.edu.pl, m.mrozowski@ieee.org

Submitted October 30, 2017 / Accepted October 31, 2017

Abstract. *This paper discusses a strategy for speeding up the mesh deformation process in the design-by-optimization of high-frequency components involving electromagnetic field simulations using the 3D finite element method (FEM). The mesh deformation is assumed to be described by a linear elasticity model of a rigid body; therefore, each time the shape of the device is changed, an auxiliary elasticity finite-element problem must be solved. In order to accomplish this in a very short time numerical integration and the solution of the resulting system of equations are performed using a graphics processing unit (GPU). The performance of the proposed algorithm is illustrated are verified using a complex example involving 3D FEM analysis of a dielectric-resonator filter.*

Keywords

Finite element method, mesh deformation, mesh morphing, GPU computing

1. Introduction

The finite-element method (FEM) is commonly used as a simulation tool in computational electromagnetics (CEM), due to its ability to solve 3D problems involving complex shapes and frequency-dependent inhomogeneous materials. Despite these advantages, the FEM suffers from high computational cost that results in part from the long time needed to mesh the structure, set up the system of equations, and subsequently solve it. Since the accuracy of simulation strongly depends on mesh quality, an iterative scheme of adaptive mesh refinement is commonly used to refine the mesh locally in areas that are crucial for the quality of the numerical solution. This procedure is time consuming; when refining the mesh locally, a series of numerical problems of increasing size needs to be solved at a single (or a few) frequency points. Each of these solutions involves factorization of a sparse linear problem. The whole procedure can thus take minutes or hours to complete.

In this respect, the problem is aggravated when the geometry of the modeled structure changes, as is the case with optimization, parametric studies, and design-sensitivity analysis. In some special cases, when the modifications are local, model order reduction techniques applied to the modified subdomains can be applied [1] to speed up the analysis. In general, however, a change in the design variables entails repeating the entire process of finding an optimal mesh. This can be avoided by using mesh deformation techniques, in which a mesh (generated during preprocessing) is modified as a result of the change in design variables by moving the internal nodes of the mesh. There are several techniques of this type [2], [3]: one involves a linear elasticity model of solid mechanics in 3D [3]. In this approach, the mesh is regarded as a rigid body and, for each perturbation of the geometry, the deformation of this body is found by solving an auxiliary finite-element mechanical problem based on Hooke's law. The advantage of this approach over alternative techniques is that it produces high-quality untangled meshes. Moreover, it is well suited for complex 3D models.

Deforming large high-order (curved) meshes by means of finite element simulations can be time-consuming. One way to reduce the time needed to deform the mesh is to make use of modern graphics processing units (GPUs) that allow concurrent execution of many computing tasks. Parallelization of the most time-consuming stages by using multicore GPU architectures has been considered for many computational techniques [4–6], including the finite-element method in [7–11]. However, it should be noted that, while modern GPUs offer higher performance in terms of theoretical peak FLOPs and bandwidth than current CPUs, each algorithm must be reformulated to make effective use of the capabilities of the GPUs. This paper investigates GPU acceleration of the mesh deformation technique discussed in [3]. In particular, we consider the generation of a global finite element matrix for linear elasticity problems, as well as the numerical solution of the system of equations that arises from this. We additionally investigate two strategies for performing defor-

mation on high-order (curved) meshes, which are often used in the finite-element analysis of electromagnetic fields.

2. Mesh Deformation

In general, the boundary-value problem (BVP) considered in CEM is defined by a governing differential equation that needs to be solved, a structure geometry (and its discretization in the form of a volumetric mesh), and boundary conditions on the structure surfaces (including excitation). When any part of the structure geometry changes, as often occurs in a parametric sweep or optimization loop, some boundaries (surfaces) move, and this information can be used to update the internal mesh and then to re-evaluate the solution on the updated mesh. The geometry may shrink, expand, be compressed, or be stretched locally, and the mesh should follow the change in the geometry. Since the geometry can be treated as a solid, the updating of the mesh-internal nodes can be organized as a solution of a linear elasticity problem with the appropriate boundary conditions. This linear elasticity problem is governed by Hooke's law

$$\sigma_{\mathrm{m}} = 2\mu_{\mathrm{m}}\epsilon_{\mathrm{m}}(u) + \lambda(\nabla \cdot u)I \tag{1}$$

where I is a 3×3 identity matrix, u is the displacement of the material particle, and μ_{m} and λ are the so-called Lamé parameters

$$\mu_{\mathrm{m}} = \frac{E_{\mathrm{m}}}{2(1+v)}, \tag{2}$$

$$\lambda = \frac{Ev}{(1+v)(1-2v)}. \tag{3}$$

Here, E_{m} is the Young's modulus and v the Poisson ratio of the material; ϵ_{m} and σ_{m} are the strain and stress symmetric tensors with six independent components. For details, see [3], [12], [13]. Applying the finite-element method to the linear elasticity equation, the following equation is obtained:

$$\int_V B^{\mathrm{T}} \cdot c \cdot B \mathrm{d}V \cdot d = 0, \tag{4}$$

where c is a symmetric matrix of material constants. For isotropic materials, c has the form

$$c = \begin{bmatrix} \lambda+2\mu_{\mathrm{m}} & \lambda & \lambda & 0 & 0 & 0 \\ \lambda & \lambda+2\mu_{\mathrm{m}} & \lambda & 0 & 0 & 0 \\ \lambda & \lambda & \lambda+2\mu_{\mathrm{m}} & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu_{\mathrm{m}} & 0 & 0 \\ 0 & 0 & 0 & 0 & \mu_{\mathrm{m}} & 0 \\ 0 & 0 & 0 & 0 & 0 & \mu_{\mathrm{m}} \end{bmatrix}, \tag{5}$$

and B is the strain matrix given by

$$B = \begin{bmatrix} \frac{\partial}{\partial x} & 0 & 0 \\ 0 & \frac{\partial}{\partial y} & 0 \\ 0 & 0 & \frac{\partial}{\partial z} \\ 0 & \frac{\partial}{\partial z} & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} & 0 & \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} & 0 \end{bmatrix} N, \tag{6}$$

with N being the nodal basis functions introduced to represent the displacement in volume V. A stiffness matrix K can be introduced:

$$K = \int_V B^{\mathrm{T}} \cdot c \cdot B \mathrm{d}V. \tag{7}$$

The unknown vector of free mesh node displacement d_f can be computed by solving the linear problem:

$$K_{ff} \cdot d_f = -K_{fc} \cdot d_c \tag{8}$$

where K_{ff} and K_{fc} are the submatrices of K associated with the free and fixed node displacements, while d_f holds the displacements of the nodes that correspond to the Dirichlet boundary condition (for details of the formulation and examples of the application of the technique to real-life engineering problems, see [3]).

3. Deformation of High-order Meshes in EM Simulations

In electromagnetic simulations using the finite element method, second-order (curved) mesh elements are usually defined only for elements lying on curved surfaces. The rest of the elements are treated as if they were rectilinear, even if mesh is of a higher order. This is done by setting the coordinates of the midpoints of the edges so that they lie on the straight lines connecting the corner nodes of the element. This allows a reduction in the cost of evaluation of a finite element matrix by applying Gaussian quadratures of a low order during the numerical integration phase of the sparse matrix generation process. On the other hand, when mesh deformation is applied to a second-order (or higher-order) mesh, all the nodes (including those corresponding to the midpoints of edges) are displaced throughout the entire volume—the whole mesh then becomes high-order, and low-order quadratures can no longer be applied. There are two ways to overcome this issue:

- Approach 1 (A1): Perform the mesh deformation as if the mesh were of first order. Since the movement of all mesh boundaries is already known, the high-order mesh information on the curved surfaces can be easily restored;
- Approach 2 (A2): Perform the mesh deformation using the high-order mesh and then later, in post-processing, straighten the mesh edges that do not belong to the curved surfaces by re-evaluating the midpoint positions of the edges.

These approaches are illustrated in Fig. 1, which shows an example of the second-order mesh deformation of a 2D mesh corresponding to the cross-section of a coaxial line. At a first glance, approach A1 seems to be more suitable, since the mechanical problem to be solved is smaller; however,

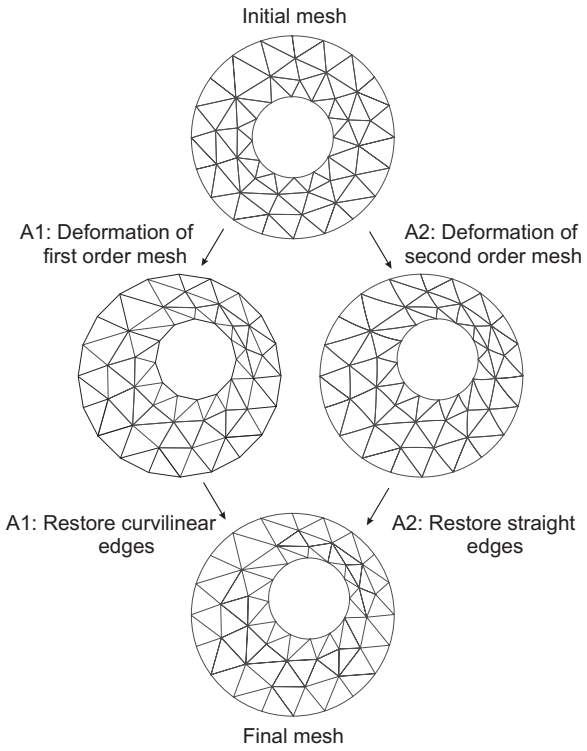

Fig. 1. Two approaches to the deformation of a high-order mesh.

the final efficiency may depend on the ratio of rectilinear to true curvilinear elements in the mesh. To gain more insight into this issue, both techniques will be further investigated; the results of comparing them will be presented and briefly discussed in Sec. 5.

4. GPU Acceleration

Regarding GPU acceleration of the mesh deformation technique from Sec. 2, it is essential to consider the following stages of the procedure:

- generation of a finite element matrix K (submatrices K_{ff} and K_{fc}) for the linear elasticity problem;
- solving the linear problem (8) with an iterative solver on a GPU.

Both stages can lead to time-consuming calculations, especially when fine meshes with a large number of elements are involved.

4.1 Generation of Finite Element Matrix

In order to generate a global finite element matrix K, the local stiffness matrices K_e must be evaluated for each element of the mesh. We here investigate the mesh deformation using four-node (linear, first-order) and ten-node (quadratic, second-order) tetrahedral elements. In the case of the first-order mesh elements (approach A1) the local matrix K_e is evaluated using closed form expressions for matrix B as

$$K_e = V_e B^{\mathrm{T}} c B. \tag{9}$$

The resulting local matrix K_e is 12×12 in size. Then, given the known mapping between the local and global degrees of freedom, the sparse matrices K_{ff} and K_{fc} can be assembled.

In the case of second-order elements (approach A2), the local matrices K_e are computed numerically using a 14-point, fourth-order Gauss quadrature rule over the tetrahedron

$$K_e = \frac{1}{6} \sum_{i=1}^{14} w_i B_i^{\mathrm{T}} c B_i dJ_i \tag{10}$$

where dJ_i is a determinant of the Jacobian matrix, w_i is the quadrature weight, and B_i is a strain matrix evaluated at the i-th point of the quadrature. In this case, B_i is a 6×30 matrix, and the resulting matrix K_e is 30×30.

Regardless of the order of the basis functions used, a basic variant of the finite element matrix generation procedure on a GPU involves the following stages:

1. preprocessing (including data allocation and transfer from a CPU to a GPU),
2. calculation of the local element matrix K_e for each mesh element,
3. construction of the K_{ff} and K_{fc} matrices in the coordinate format (COO),
4. conversion of K_{ff} and K_{fc} from COO to CRS matrix representation format
5. postprocessing (which includes the transfer of the sparse matrices from a GPU to a CPU).

Massive parallelization of the computations occurs in stages (2)–(4), however, stages (1) and (5) cannot be omitted from a model in which a GPU is used as a coprocessor for a CPU. What differentiates approaches (A1) and (A2) is the implementation of stage (2):

- Case A1: Two levels of parallelization of the computations can be proposed. First, the dense matrix–matrix computations are performed in parallel, with one GPU thread employed to perform the computations on one row. Second, the K_e matrices are calculated independently (simultaneously) for each group of elements.
- Case A2: Here, a Gaussian quadrature is used to evaluate K_e and the computations are parallelized on three levels: the dense matrix–matrix computations are performed in parallel, the computations for each quadrature point are performed in parallel, and the K_e matrices are calculated independently (simultaneously) for each group of elements.

For relatively small finite element meshes (where all the data fit on a single GPU), the matrix-assembly process can be executed in stages (3) and (4). For larger matrices, an iterative variant of finite element matrix generation algorithm [14]

must be applied. With the iterative technique, all mesh elements are divided into M smaller subsets of elements; then, in the m-th iteration, stages (2)–(4) are performed to construct the sparse submatrices K_{ff}^{m} and K_{fc}^{m}, which are then successively transferred into CPU RAM and summed. This iterative process leads to the construction of the global coefficient sparse matrices $K_{ff} = K_{ff}^{1} + \ldots + K_{ff}^{M}$ and $K_{fc} = K_{fc}^{1} + \ldots + K_{fc}^{M}$.

4.2 Iterative Solution of Linear Problems on GPUs

The global finite element matrix K_{ff} arising from linear elasticity problems is large, sparse, and positive-definite. A natural choice for the solution method of such a linear problem (8) is thus the conjugate gradient (CG) algorithm. In this paper, we apply the CG algorithm with a simple Jacobi preconditioner. This technique is especially well suited to GPU acceleration, since the main stage of the CG algorithm is a sparse matrix–vector product that can be executed on a GPU with high efficiency. To obtain good performance, GPUs need unconventional sparse matrix storage formats. The Sliced ELLR-T format intended for calculating SpMV products on a GPU is used here [15]. Computation of the BLAS1 vector operations that occur in CG algorithm were executed on a GPU. The programming model requires the allocation of CG vectors to GPU memory, the transfer of a sparse matrix (K_{fc}), and a right-hand vector ($r = K_{fc} \cdot d_c$) to a GPU, with the execution of the CG iterative process on a GPU until a solution is found with the assumed tolerance. Once convergence has been reached, the solution vector is transferred from a GPU to a CPU.

Our main interest is mesh deformation; the solution accuracy conditions can thus be relaxed for mechanical problems—in fact, a coarse approximation of mesh node displacements will suffice. In this context, an iterative solver is even preferred over a direct solver, as low precision results can be achieved relatively fast and with low memory costs.

5. Example: Sixth-order Dielectric Resonator Filter

The GPU acceleration algorithms were validated on a real-life example involving deformation of a tetrahedral mesh stemming from the finite-element electromagnetic optimization of a sixth-order dielectric resonator filter [16]. The filter was optimized using a 3D FEM CAD framework [17], and we investigated the possibilities of reducing the overall runtime. A 3D view of the structure geometry is shown in Fig. 2. The structure consists of high permittivity dielectric resonators, operating with of the $TE_{01\delta}$ mode, cascaded along an evanescent mode waveguide; the resonators are coupled by evanescent modes to achieve the cross-couplings between nonadjacent resonators, leading to two transmission zeros in

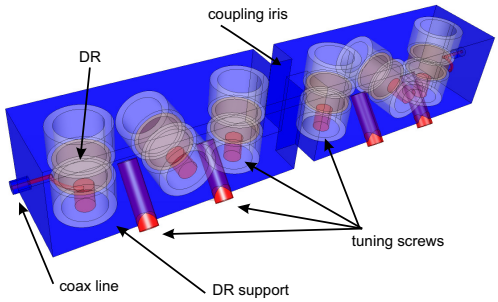

Fig. 2. 3D view of the dielectric resonator filter.

the transmission characteristic. The filter is extremely narrowband, and its numerical solution is thus sensitive to the mesh. In fact, for this example, numerical design tuning can be performed efficiently only by applying mesh deformation techniques. In this structure, as many as 18 design variables representing structure dimensions were defined: waveguide lengths, resonator spacings, tuning screw depth, iris width, and source/load to resonator coupling were controlled during the optimization process.

An optimal mesh was produced at the initial stage by an iterative process involving local error indicators. From now on, the mesh deformation was applied to transform the mesh whenever the geometry was modified. All the design variables were allowed to change at the same time. The mesh consisted of 776,306 tetrahedral elements, and the resulting matrix sizes are shown in Tab. 1. Second-order (curvilinear) tetrahedral elements were used to accurately represent the curved surfaces. Both mesh deformation approaches (A1 and A2) mentioned in Sec. 3 were tested. The linear elasticity problem to be solved for approach A1 involves 290,000 unknowns, compared to 2.7 million for approach A2.

An in-house, multithreaded, CPU-only implementation was used as reference code; this was based on Intel MKL libraries optimized for Intel architectures. For a GPU implementation we used CUDA. All numerical tests were executed on an Intel Xeon (E5-2680 v3, 2.5 GHz) with 256 GB memory and a Pascal P100 accelerator. Table 2 shows the time taken by the two main stages of CPU-based and GPU-accelerated mesh deformation when an iterative solver was used in both implementations. It can be seen that, for approach A1, the runtime of the matrix generation and solution stages are almost equal on a CPU. For approach A2 on a CPU, the solution stage dominates the total mesh deformation process. The advantage of the proposed GPU-accelerated approach over

Matrix	A1	A2
K_{ff} rows	291,246	2,741,652
K_{ff} cols	291,246	2,741,652
K_{ff} nnz	12,035,844	211,565,844
K_{fc} rows	291,246	2,741,652
K_{fc} cols	105,579	429,489
K_{fc} nnz	1,394,613	19,710,576

Tab. 1. Test problem properties.

Approach	A1	A1		A2	A2	
Device	CPU	GPU	GPU vs.	CPU	GPU	GPU vs.
Stage/ Time	[s]	[s]	CPU	[s]	[s]	CPU
MatGen	**3.57**	**4.27**	**0.8**	**30.19**	**10.80**	**2.8**
Integration	1.06	0.26	4.1	6.6	1.44	4.6
Assembly	2.51	0.42	5.9	23.59	3.48	6.8
Pre&Post Proc.	—	3.59	—	—	5.88	—
Solution	**3.10**	**0.26**	**11.9**	**95.08**	**8.57**	**11.1**
Direct/PCG	3.10	0.09	32.9	95.08	6.22	11.1
Pre&Post Proc.	—	0.17	—	—	2.36	—
Total	**6.67**	**4.53**	**1.5**	**125.27**	**19.37**	**6.5**

Tab. 2. Comparison of GPU-based mesh deformation versus CPU-reference implementation. In solution stage, the direct solver and iterative solver (PCG with residual tolerance = 10^{-7}) is used on a CPU and on a GPU, respectively.

Approach	A1	A1	A2	A2
ϵ	mean	max	mean	max
10^{-7}	$1.7 \cdot 10^{-8}$	$2.4 \cdot 10^{-7}$	$5.3 \cdot 10^{-8}$	$1.0 \cdot 10^{-6}$
10^{-6}	$1.9 \cdot 10^{-7}$	$2.8 \cdot 10^{-6}$	$4.8 \cdot 10^{-7}$	$8.9 \cdot 10^{-6}$
10^{-5}	$1.8 \cdot 10^{-6}$	$4.3 \cdot 10^{-5}$	$4.4 \cdot 10^{-6}$	$9.5 \cdot 10^{-5}$
10^{-4}	$1.9 \cdot 10^{-5}$	$3.2 \cdot 10^{-4}$	$4.7 \cdot 10^{-5}$	$1.1 \cdot 10^{-3}$

Tab. 3. Mean and maximum errors of displacement d_f when Eq. (8) is solved with direct and iterative solvers for different residual tolerances (ϵ).

the CPU-based reference implementation can immediately be seen by considering the time taken for numerical integration, matrix assembly, and solution of a sparse system, which were over 4, 6, and 11 times shorter. It can be seen that using a GPU results in significant overhead due to preprocessing and postprocessing related to data allocation on the GPU and transfers between the GPU and CPU. As a result, the approach A1 computations on a GPU are about 1.5 times faster than a CPU-only implementation using a direct solver (which is about the same what one gets with a CPU-only implementation and an iterative solver). For approach A2, however, where the generated sparse system is significantly larger than in the A1 approach, use of the GPU results in 6-fold speedup.

The influence of the residual tolerance of the iterative solver (PCG) on the performance and accuracy of the mesh deformation techniques was also investigated. Table 3 compares the mean and maximal errors of the displacement vector d_f (Eq. (8)) obtained with direct and iterative solvers. The direct solution of a sparse system of linear equations was carried out on a CPU using Intel Pardiso. For the iterative solver, various residual tolerances ϵ were considered. It can be seen that relaxing the tolerance affects the quality of the mesh deformation; however, even for $\epsilon = 10^{-4}$, the accuracy is satisfactory (d_f contains the displacements of the nodes as $\Delta x, \Delta y, \Delta z$ in meters). As expected, Tab. 4 shows that an iterative solver is faster than a direct solver, regardless of whether it is executed on a CPU or a GPU; the slackening of the residual tolerance bears fruit in the noticeable reduction in the time taken by the solution stage. Taking into account all the stages and the preprocessing required by the GPU, we obtain the results given in Tab. 5. It can be concluded that the overall performance of the entire mesh deformation process is significantly better when a linear system is solved with the PCG algorithm. However, due to the overhead, it pays off to use a GPU only for the second approach (A2). In this case,

Approach	A1	A2	A1	A2
ϵ	PCG(CPU) vs. Direct	PCG(CPU) vs. Direct	PCG(GPU) vs. Direct	PCG(GPU) vs. Direct
10^{-7}	3.5	3.0	11.9	11.1
10^{-6}	4.1	3.5	12.5	12.6
10^{-5}	5.0	3.6	12.6	13.5
10^{-4}	6.2	3.8	11.5	16.9

Tab. 4. Speedup of CPU-based and GPU-based iterative solver vs. direct solver on CPU with changing residual tolerance (ϵ).

MatGen	Solve	A1 [s]	Speedup	A2 [s]	Speedup
CPU	CPU,Direct	6.7	1.0	125.3	1.0
CPU	CPU,10^{-7}	4.4	1.5	62.2	2.0
CPU	CPU,10^{-6}	4.3	1.5	57.1	2.2
CPU	CPU,10^{-5}	4.2	1.6	56.6	2.2
CPU	CPU,10^{-4}	4.1	1.6	55.4	2.3
GPU	GPU,10^{-7}	4.5	1.5	19.4	6.5
GPU	GPU,10^{-6}	4.5	1.5	18.3	6.8
GPU	GPU,10^{-5}	4.4	1.5	17.8	7.0
GPU	GPU,10^{-4}	4.4	1.5	16.4	7.6

Tab. 5. Comparison of implementations of two mesh deformation approaches according to the device that performs the computations and residual tolerance of the iterative solver PCG with Jacobi preconditioner.

the mesh deformation is seven times faster that when the deformation is carried out on a CPU with a direct solver. For approach A1, both CPU and GPU perform equally well and provide a 1.5-fold acceleration over a CPU implementation involving the factorization-based direct solution of a linear system.

6. Conclusions

We have shown that 3D mesh deformation in the finite element method can be significantly accelerated when a graphics processing unit is employed to perform computations. We have also described the strategies for performing deformation of higher-order meshes targeted to electromagnetic field simulations with FEM.

Acknowledgments

This work was supported by the Polish National Science Centre under contract UMO-2013/09/B/ST7/04202. A Pascal P100 graphics accelerator was purchased from the funds provided by the Faculty of Electronics, Telecommunications, and Informatics at Gdańsk University of Technology. Technical support from EM Invent [18] is gratefully acknowledged.

References

[1] FOTYGA, G., NYKA, K. Efficient analysis of structures with rotatable elements using model order reduction. *Radioengineering*, 2016, vol. 25, no. 1, p. 73–80. DOI: 10.13164/re.2016.0073

[2] STATEN, M. L., OWEN, S. J., SHONTZ, S. M., et al. A comparison of mesh morphing methods for 3D shape optimization. In *Proceedings of the 20th International Meshing Roundtable*. 2012, p. 293–311. DOI: 10.1007/978-3-642-24734-7_16

[3] LAMECKI, A. A mesh deformation technique based on solid mechanics for parametric analysis of high-frequency devices with 3-D FEM. *IEEE Transactions on Microwave Theory and Techniques*, 2016, vol. 64, no. 11, p. 3400–3408. DOI: 10.1109/TMTT.2016.2605672

[4] ZYGIRIDIS, T. T. High-order error-optimized FDTD algorithm with GPU implementation. *IEEE Transactions on Magnetics*, 2013, vol. 49, no. 5, p. 1809–1812. DOI: 10.1109/TMAG.2013.2241410

[5] LIVESEY, M., STACK, J. F., COSTEN, F., et al. Development of a CUDA implementation of the 3D FDTD method. *IEEE Antennas and Propagation Magazine*, 2012, vol. 54, no. 5, p. 186–195. DOI: 10.1109/MAP.2012.6348145

[6] MU, X., ZHOU, H.-X., CHEN, K., et al. Higher order method of moments with a parallel out-of-core LU solver on GPU/CPU platform. *IEEE Antennas and Propagation Magazine*, 2014, vol. 62, no. 11, p. 5634–5646. DOI: 10.1109/TAP.2014.2350536

[7] ZHANG, J., SHEN, D. GPU-based implementation of finite element method for elasticity using CUDA. In *Proceedings of the IEEE 10th International Conference on High Performance Computing and Communications and IEEE International Conference on Embedded and Ubiquitous Computing*. 2013, p. 1003–1008. DOI: 10.1109/HPCC.and.EUC.2013.142

[8] MENG, H. T., NIE, B. L., WONG, S., et al. GPU accelerated finite-element computation for electromagnetic analysis. *IEEE Antennas and Propagation Magazine*, 2014, vol. 56, no. 2, p. 39–62. DOI: 10.1109/MAP.2014.6837065

[9] DINH, Q., MARECHAL, Y. Toward real-time finite-element simulation on GPU. *IEEE Transactions on Magnetics*, 2016, vol. 52, no. 3, p. 1–4. DOI: 10.1109/TMAG.2015.2477602

[10] GUAN, J., YAN, S., JIN, J. M. An accurate and efficient finite element-boundary integral method with GPU acceleration for 3-D electromagnetic analysis. *IEEE Transactions on Antennas and Propagation*, 2014, vol. 62, no. 12, p. 6325–6336. DOI: 10.1109/TAP.2014.2361896

[11] AKINCI, G., YILMAZ, A. E., KUZUOGLU, M. Excessive memory usage of the ELLPACK sparse matrix storage scheme throughout the finite element computations. *Radioengineering*, 2014, vol. 23, no. 4, p. 997–1004. ISSN: 1210-2512

[12] LIU, G. R., QUEK, S. S. *The Finite Element Method: A Practical Course*. 1st ed. Butterworth–Heinemann, 2003. ISBN: 9780750658669

[13] COOK, R. D. *Finite Element Modelling for Stress Analysis*. John Wiley & Sons, 1995. ISBN: 978-0471107743

[14] DZIEKONSKI, A., SYPEK, P., LAMECKI, A., et al. Generation of large finite-element matrices on multiple graphics processors. *International Journal for Numerical Methods in Engineering*, 2013, vol. 94, no. 2, p. 204–220. DOI: 10.1002/nme.4452

[15] DZIEKONSKI, A., LAMECKI, A., MROZOWSKI, M. Tuning a hybrid GPU-CPU V-cycle multilevel preconditioner for solving large real and complex systems of FEM equations. *IEEE Antennas and Wireless Propagation Letters*, 2011, vol. 10, p. 619–622. DOI: 10.1109/LAWP.2011.2159769

[16] BASTIOLI, S., SNYDER, R. V. Inline pseudoelliptic $TE_{01\delta}$-mode dielectric resonator filters using multiple evanescent modes to selectively bypass orthogonal resonators. *IEEE Transactions on Microwave Theory and Techniques*, 2012, vol. 60, no. 12, p. 3988–4001. DOI: 10.1109/TMTT.2012.2222659

[17] LAMECKI, A., BALEWSKI, L., MROZOWSKI, M. An efficient framework for fast computer aided design of microwave circuits based on the higher-order 3D finite-element method. *Radioengineering*, 2014, vol. 23, p. 970–978. ISSN: 1210-2512

[18] EM INVENT, POLAND. *Solutions for RF, Microwave Engineering and Computational Electromagnetics*. [Online] Cited 2017-10-30. Available at: www.eminvent.com

About the Authors . . .

Adam LAMECKI received M.Sc. and Ph.D. (with honors) degrees in microwave engineering from Gdańsk University of Technology (GUT), Gdańsk, Poland, in 2002 and 2007, respectively. He was the recipient of a Domestic Grant for Young Scientists awarded by the Foundation for Polish Science in 2006. In 2008, he received the Prime Minister's Award for his doctoral thesis and, in 2011, a scholarship from the Ministry of Science and Higher Education. His research interests include surrogate models and their application in the CAD of microwave devices, computational electromagnetics (mainly focused on the finite element method), and filter design and optimization techniques.

Adam DZIEKONSKI received M.S.E.E. and Ph.D. degrees (with hons.) in microwave engineering from Gdańsk University of Technology, Gdańsk, Poland, in 2009 and 2015, respectively. His current research interests include computational electromagnetics (mainly focused on the parallelizing computations on graphics processing units and central processing units). Dr. Dziekonski has twice been a recipient of the Domestic Grant for Young Scientists from the Foundation for Polish Science in 2012 and 2013. He was also a recipient of the Prime Minister's Award for his doctoral thesis in 2016.

Lukasz BALEWSKI received M.Sc. and Ph.D. (with honors) degrees in microwave engineering from Gdańsk University of Technology (GUT), Gdańsk, Poland, in 2003 and 2008, respectively. His research interests include CAD of microwave devices, filter design, and optimization techniques. He is the coauthor of several software tools for microwave filter design.

Grzegorz FOTYGA received M.S.E.E. and Ph.D. degrees in electronic engineering from Gdańsk University of Technology in 2009 and 2016, respectively. He is currently an assistant professor with the Department of Microwave and Antenna Engineering, Gdańsk University of Technology. His current research interests include computational electromagnetics, numerical methods, the finite element method, and model order reduction.

Michal MROZOWSKI received M.Sc. and Ph.D. degrees, both with honors, from Gdańsk University of Technology in 1983 and 1990, respectively. In 1986, he joined the Faculty of Electronics, Gdańsk University of Technology, where he is now a Full Professor, Head of the Department of Microwave and Antenna Engineering, Director of the Center of Excellence for Wireless Communication Engineering (WiComm). He is a Fellow of IEEE.