Accelerating Multi GPU Based Discontinuous Galerkin FEM Computations for Electromagnetic Radio Frequency Problems

Nico Gödel [1], Nigel Nunn, Tim Warburton [2], and Markus Clemens [3]

[1] Faculty of Electrical Engineering
Helmut-Schmidt-University, University of the Federal Armed Forces Hamburg,
P.O. Box 700822, D-22008 Hamburg, Germany
Nico.Goedel@hsu-hh.de

[2] Computational and Applied Mathematics
Rice University, 6100 Main Street MS-134, Houston, TX, USA

[3] Bergische Universität Wuppertal, FB E, Chair for Electromagnetic Theory
Rainer-Gruenter-Str. 21, D-42119 Wuppertal, Germany

***Abstract*—**A Graphics Processing Unit (GPU) accelerated simulation of Maxwell's equations in the time domain is presented. The Discontinuous Galerkin Finite Element Method (DG-FEM) is used for discretization since the elementwise structure fits the parallelization design aspects of the GPU architecture and the NVIDIA Compute Unified Device Architecture (CUDA), a GPU programming model. The parallelization strategy for a multi-GPU setup using CUDA is focused. Several performance improvements are analyzed and investigated with the help of a realistic 3D electromagnetic scattering example.

***Index Terms*—**GPU-Computing, GPGPU, DG-FEM, electromagnetics, CUDA, TESLA.

I. INTRODUCTION

Numerical simulation of electromagnetic devices during the development and certification process can significantly reduce time, efforts and costs. Efficiency and costs of numerical simulations depend on hardware and software investments as well as on personnel expenses, which directly evolve from code performance and simulation time. The presented hardware accelerated approach is able to significantly reduce both, simulation time and hardware costs using consumer based GPUs instead of highly expensive large scale computing clusters.

Hardware accelerated computation is not a new research domain, but recently gained attention due to the availability of high-level compute abstractions such as CUDA [1], BROOK+ [2] and OPENCL [3]. Furthermore, floating-point performance and device memory bandwidth of current consumer based GPUs exceed their CPU counterparts by more than one order of magnitude at approximately the same price per unit.

The combination of these GPU based co-processing units and the evolving programming models provide a significant potential for high-performance related computations.

This potential has been investigated for different volume based discretization methods, such as the Finite Difference Method [4, 5] and the Finite Integration Technique. Recently, the Discontinuous Galerkin Finite Element Method (DG-FEM) has gained attention in connection with GPU computations [6, 7].

In this paper, the parallelization model and several optimization techniques for DG-FEM computations on GPU-clusters will be investigated. The focus will be on the scalability of multi-GPU systems.

The paper is organized as follows: In Section II, the model is defined stating both the governing differential equations for the electric and the magnetic field and the spatial discretization using DG-FEM. Subsequently, in Section III, the DG-FEM discretization is investigated with respect to the suitability of a parallel implementation. In

Section IV, the parallelization model for three different abstraction layers as well as the hardware in use is presented and Section V provides numerical results for the different types of implementations with the use of a complex example.

II. DESCRIPTION OF THE MODEL

A. Maxwell's Equations

Electromagnetic wave propagation in lossless medium can be described using Ampère's law together with Faraday's law of induction

$$\frac{\partial}{\partial t}\varepsilon\vec{E} = \nabla \times \vec{H} \quad (1)$$

$$\frac{\partial}{\partial t}\mu\vec{H} = -\nabla \times \vec{E} \quad . \quad (2)$$

Here, $\vec{E}$ and $\vec{H}$ denote the electric and magnetic field strength, respectively, whereas ε and μ identify the electric permittivity and the magnetic permeability. The right-hand sides (RHS) of (1) and (2) describing the spatial dependencies can be discretized with help of the Nodal Discontinuous Galerkin method.

B. Discontinuous Galerkin Discretization

DG-FEM was first introduced by Reed and Hill in 1973 [8] for neutron transport simulation and during the last decade, this method was intensively investigated for solving Maxwell's equations. Relevant results have been published especially by Cockburn et. al. [9], Cohen et. al. [10] and by Hesthaven and Warburton [11, 12].

The DG method was chosen due to some important characteristics, i.e.
(1) the treatment of complex geometry through unstructured tetrahedral meshes,
(2) explicit time stepping schemes, especially multirate time stepping schemes,
(3) the use of high order basis functions and
(4) a domain decomposition approach, which is intrinsically included in the DG formulation.

Along with these features, the caveats are restricted to geometry-dependent time steps and an overhead in degrees of freedom at the element faces compared to continuous FEM as well as not providing a strictly conservative model of electric charges.

A Nodal DG discretization of eqn. (1) and (2) is derived in [12] using Lagrange polynomials as basis functions. The main characteristic of DG-FEM is that it allows for an elementwise computation of the elements, i.e. each element is computed separately. The semi-discrete scheme, still continuous in time, for each element reads

$$\frac{\mathrm{d}}{\mathrm{d}t}\varepsilon\mathbf{E} = \mathbf{M}^{-1}\mathbf{S}\,\mathbf{H} - \mathbf{M}^{-1}\mathbf{F}\cdot[\hat{\mathbf{n}}\cdot(\mathbf{f}_{\mathsf{E}} - \mathbf{f}^{*}_{\mathsf{E}})] \quad (3)$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\mu\mathbf{H} = -\mathbf{M}^{-1}\mathbf{S}\,\mathbf{E} + \mathbf{M}^{-1}\mathbf{F}\cdot[\hat{\mathbf{n}}\cdot(\mathbf{f}_{\mathsf{H}} - \mathbf{f}^{*}_{\mathsf{H}})] \quad . (4)$$

Here, $\mathbf{M}^{-1}$ is the inverse of a local mass-matrix, **S** a local stiffness-matrix and **F** a local flux-matrix. The size of the matrices depends on the number of nodes inside each element. The size of the symmetric matrices **M** and **S** is listed in Table 1 for different polynomial orders. The term $\mathbf{M}^{-1}\mathbf{S}$ refers to a local differentiation in the element without the need for using a custom elementwise mass-matrix for every single element.

The flux-matrix **F** refers to the integration over every triangular face of each tetrahedron. The size of the flux matrix **F** depends on the number of nodes inside the element and the number of nodes on all surfaces of the element. The flux terms f_E and f_H in eqn. (3) and (4) refer to flux density terms of adjacent face values on each face.

III. PARALLEL STRUCTURE OF THE MODEL

To allow for efficient parallelization on GPUs as well as on CPUs, the computation has to be split up into small pieces of work. Ideally, each piece of work has a completely independent data structure, thus requires no memory interaction with other parts. However, treating hyperbolic problems, it seems clear that there has to be communication, at least for neighboring elements, to resolve the propagation of electromagnetic waves. The idea of the DG-FEM and the proposed implementation is to minimize and encapsulate all dependencies and to efficiently handle communication with help of special GPU features.

As described in section II.B., the first operation of the RHS, referring to the curl computation on Maxwell's equation, is computed inside each element. Here, every element can be computed completely independent from each other, providing the opportunity of a highly parallel implementation. The second term in the

RHS has to be treated more carefully. In the flux term evaluations, the jumps in the electric and magnetic fields are integrated over the elements faces. The integration itself, being numerically expensive, is computed for each element separately. The jump computation however needs data from adjacent elements. This computation has to be treated in a special way regarding elementwise, highly parallel implementations and will be presented in Section IV.C.

IV. HARDWARE, PROGRAMMING MODEL AND PARALLELIZATION STRATEGY

A. TESLA S1070 GPU Server

For the GPU computations, a NVIDIA TESLA S1070 with four GPUs and 4 GB GDDR3 RAM each is used. Each GPUs consists of 30 Multiprocessors (MP) with 8 Streamingprocessors (SP) each. The GPU server is attached to the host server using two PCIe x16 Gen2 connections.

B. Coarse Grained Parallelism

Regarding GPU-cluster computations, the most coarse grained parallelization can be realized using a METIS [13] domain decomposition of the computational domain as presented in Fig. 1.

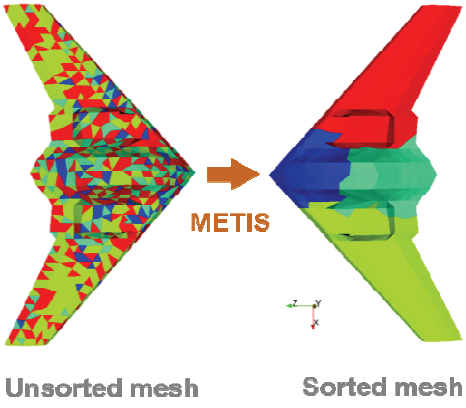

Fig. 1. Domain decomposition using METIS graph partitioning.

Each color maps the tetrahedral elements to one GPU. The METIS domain decomposition can be executed using two constraints: load balancing and minimization of communication between the subdomains. In [7], an algorithm using KMETIS was presented, providing good minimization of the surface, but not optimal load balancing for this small amount of subdomains. In this work, PMETIS being more suitable for a small amount of subdomains is used. The resulting performance improvements are presented in Section V.

Each METIS subdomain is computed on one GPU. For the flux computation presented in Section III, field data of neighboring elements across METIS boundaries have to be provided to the corresponding GPU. Since this communication has to be carried out as a CPU task via the PCIe bus, the data exchange is supposed to be a potential bottleneck of the presented approach. To minimize the effects of the latency and the bandwidth of the PCIe bus, two aspects should be ensured:

1. Only the data which are needed by another GPU should be transferred, and only to this specific device.
2. The data transfer should be hidden behind other computations with help of asynchronous file transfer.

The second part is one of the key aspects regarding highly scalable code on multi-GPU systems. On NVIDIA TESLA GPU devices with compute capability 1.1, the possibility of simultaneous execution of kernel functions and host-device/device-host memory transfer is introduced. A kernel can be executed concurrently to a data transfer to or from the host. This is applicable as long as the kernel does not depend on the transferred data. Regarding the presented DG-FEM implementation, the kernel function evaluating the curl operator inside each element can be executed while field data of adjacent METIS subdomain boundaries is transferred. With this feature, the scalability bottleneck can be expanded.

C. GPU Based Block Parallelism

On each GPU, one METIS subdomain is computed. All data (fields, geometry and operators) related to this subdomain are stored in the device memory and stays in the device memory for the entire simulation. Using CUDA as a programming model, the computational work is arranged in a CUDA GRID. This GRID consists of CUDA BLOCKS, each having the same amount of computational work. This data management is

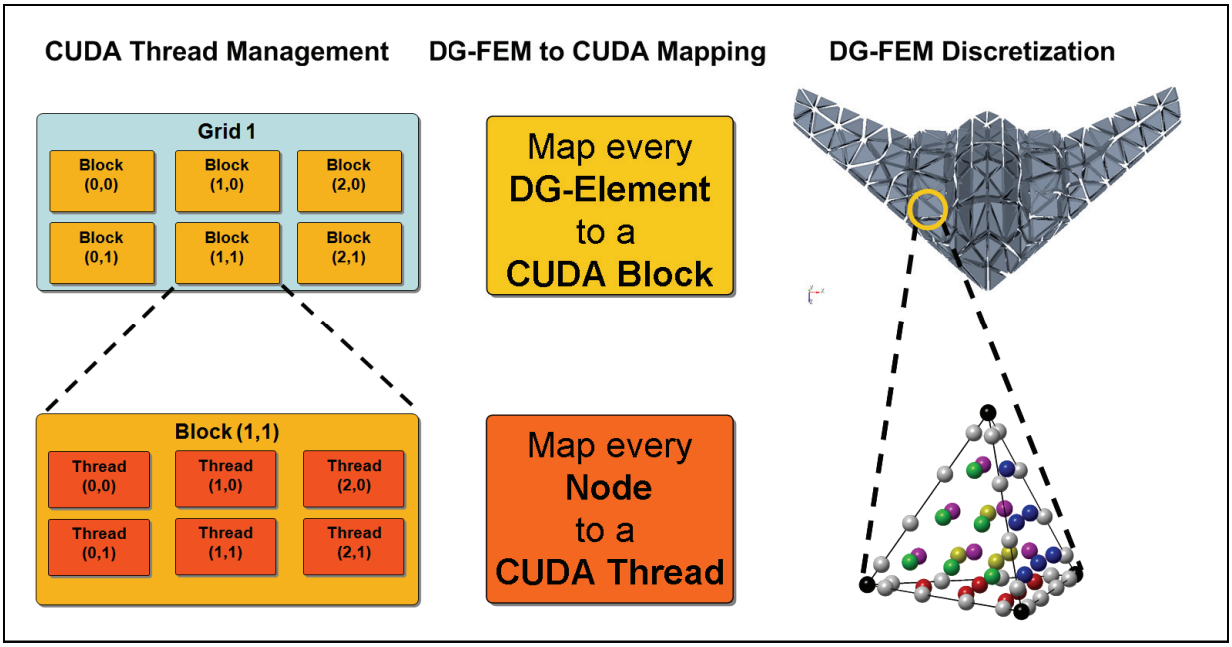

Fig. 2. Correlation between DG-FEM discretization and CUDA data management.

pictured in Fig. 2, highlighting the proposed strategy of implementing DG simulations in CUDA. Instead of conventional vector based implementations, branching is possible within this Single Instruction Multiple Thread (SIMT) architecture. CUDA BLOCKS cannot communicate with each other during their execution. One BLOCK is executed on a MP, where it profits from the high-speed, low-latency shared memory space within each MP. Global memory fetches have to be executed in a coalesced way, avoiding multiple read operations on a single address. In the case that coalesced reads cannot be ensured, read conflicts are serialized with the effect of several hundreds of cycles latency for each conflict. In this case, the use of texture memory is beneficial, providing buffered memory access at almost the same bandwidth as coalesced global memory fetches.

As long as all operations are done locally within each element, coalesced reads can be ensured for the fields. However, when calculating the curl operator in the RHS, spatial derivatives in the reference element have to be provided for all the elements. Here, coalesced reads cannot be ensured and the use of texture buffered memory access provides higher performance as presented in section V.

Regarding flux computations, for the evaluation of field differences of the triangular faces, each field value at the faces will be read more than one time, leading to serialized memory fetches. This read conflict is less severe than the one earlier presented since the number of read conflicts depends on the maximum number of elements, a vertex is connected to. However, also in this case, the use of texture memory is preferable.

Figure 2 highlights that the GRID – BLOCK decomposition within the CUDA model is reflecting the geometric discretization of each METIS subdomain with help of finite elements, here tetrahedral elements. The idea of the proposed approach is to map every finite element to a CUDA BLOCK.

D. Fine Grained GPU Thread Based Parallelism

The lowest level of abstraction within the parallelization strategy is formed by the threads managed by each MP for the computation of each BLOCK. Each MP is able to manage several hundreds of threads at the same time to feed the 8 SP. The instruction unit of each MP can process one instruction every 4 cycles. Therefore a set of

32 CUDA THREADS form a WARP, the smallest scheduling unit.

Since the solution in each DG element will be approximated with nodal basis functions, the nodes inside each element reflect the degrees of freedom for each field component. Fig. 2 highlights the proposed strategy, which maps the nodes, i.e. the degrees of freedom inside each element to the CUDA THREADS. The number of nodes Np depends on the order N of the polynomial basis functions with Np = (N+1)(N+2)(N+3)/6 (see Table 1). Since the number of nodes does not match a multiple of a single WARP, some THREADS do not contribute to the computation. A strategy for improving the efficiency using these "padding" threads can be found in [6], where several element are grouped together for a low polynomial order N.

Table 1: Polynomial order and number of nodes.

N	Np
1	4
2	10
3	20
4	35
5	56
6	84
7	120
8	165

In the code presented in this work, the number of threads per CUDA BLOCK is defined as the number of nodes in one element.

V. NUMERICAL INVESTIGATIONS

In this section, the effects of several CUDA related optimization techniques are investigated. The accuracy of the presented approach is presented in [7]. In this work, more challenging geometries are considered. As application, an electromagnetic scattering object as presented in Figs. 1 and 2 is used.

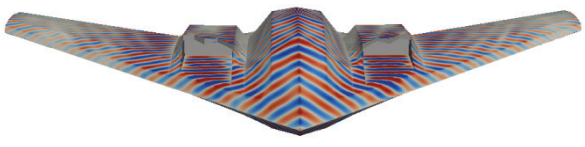

Fig. 3. Ey surface field component of a scatterer hit by a TEM wave.

A TEM wave with 0.5 GHz is used as excitation function. The electrical surface field component in y-direction is presented in Fig. 3. The object is treated as perfect electric conductor and is surrounded by vacuum medium which is enclosed by an absorbing layer. The domain is discretized with 143936 tetrahedra. Unless otherwise noted, the simulations were carried out using 6^{th} order polynomials leading to $7.2 \cdot 10^7$ unknowns and 5362 timesteps using the Low Storage Explicit Runge-Kutta (LSERK) scheme. On a single GPU, 2.02 GB memory is needed for this configuration.

A. Texture Buffered Memory Fetches

In this subsection, the effects of texture buffered memory fetches are investigated. As described in Section IV.C, the use of texture memory can be beneficial whenever coalesced reads cannot be ensured. The most drastic performance increase has been encountered within the curl computation of (3) and (4). Here, local derivatives in the reference element have to be provided for all elements / CUDA BLOCKS. The presented approach uses texture memory to buffer the operator and shared memory to buffer the field data.

Within the flux computation, the evaluation of the field differences at the faces cannot be realized coalesced, however, the number of conflicts is small. Here, the caching of field data through texture memory is analyzed.

On a single GPU, using texture fetches for the derivatives in the curl computation yields a performance improvement of a factor of 2.69, as presented in Table 2.

Table 2: Performance gain with help of texture buffered memory fetches on a single GPU.

Implementation	Performance [GFlops]	Speedup
Without TEXTURE usage	79.3	1.0
TEXTURE usage for curl computation	213.2	2.69
TEXTURE usage for flux computation	78.9	0.995
TEXTURE usage for curl and flux computation	209.3	2.64

For the flux computation, the expected performance improvement cannot be produced. In contrast, texture buffered memory fetches are slightly less efficient than their global memory counterparts. In comparison to the former G92 core, where performance increases of 6% were encountered, NVIDIA seems to have improved the global memory access. In [1], the constraints for using coalesced memory access are weakened compared to earlier documentations, which might be one reason for the observed changes in different architectures.

To summarize, the use of the texture units for buffered memory access can speed up the implementation whenever a multiplicity of read conflicts occur. In case that the number of read conflicts is small compared to the whole data volume transferred, global memory fetches can profit from their larger bandwidth.

B. Performance and Scalability of Multi-GPU Computations

In this subsection, performance and scalability of multi-GPU computations using the four GPUs of a TESLA S1070 server are investigated. CPU computations were carried out on four AMD quad-core Opteron CPUs with 2.3 GHz. The price for the S1070 is about 2900€ (academic pricing) compared to 16.759€ of the latest HP Proliant DL785G CPU server.

In Fig. 4, a comparison of GPU and CPU performance for different polynomial orders is presented.

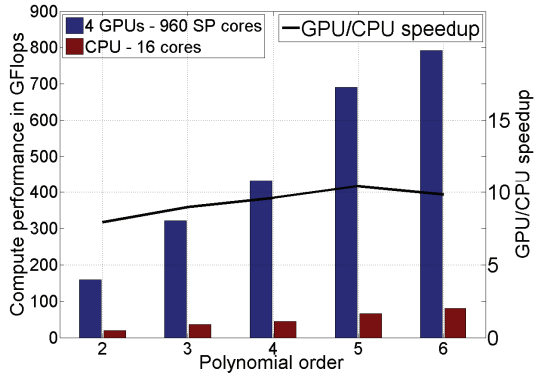

Fig. 4. Performance of GPU and CPU computations for different polynomial orders.

All computations were carried out using the IEEE-754 single precision floating point standard. GPU performance is about ten times higher than the corresponding CPU implementation.

In Fig. 5, the scalability of multi-GPU computations is presented using different METIS distributions. Furthermore, the effect of asynchronous file transfer is highlighted. For the scalability evaluation, a polynomial order of 5 has been chosen. With help of the perfectly load balanced PMETIS distribution and asynchronous file transfer, a strong scalability of 98.8 % is achieved. Due to the asynchronous file transfer, almost the complete communication overhead could be hidden behind the arithmetic curl computation which needed 15ms time in average in contrast to the communication which required 3ms in average. Except for a numerically cheap packaging of the transferred data, the presented solution incorporated zero communication overhead. The difference in parallel communication and curl computation time yields potential for further parallelization with eight GPUs at the same high degree of scalability.

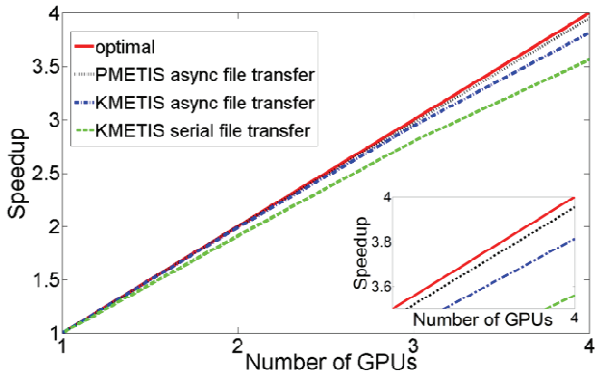

Fig. 5. Scalability of multi-GPU DG computations.

Strong scalability in this case means that the global problem size does not change when increasing the number of GPUs. However, according to [6], a minimum work of approximatively 10000 elements per GPU should be provided to get the full floating point performance. With the 143936 elements of the scattering example, the problem could be distributed on more GPUs without losing much of the efficiency.

The METIS distributions are presented in Fig. 6 highlighting the difference in load balancing for the four subdomains.

The PMETIS distribution is perfectly balanced whereas the KMETIS distribution leads to unbalanced workload, where thread number 2 has to do more work than the other threads.

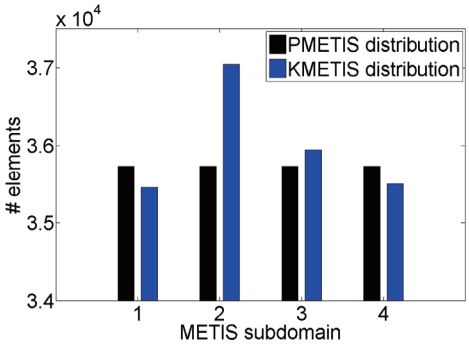

Fig. 6. Partitioning of PMETIS and KMETIS algorithms for four subdomains.

VI. CONCLUSION

A CUDA based GPU implementation of Maxwell's equations in the time domain was presented. The matching modeling principles of DG-FEM and CUDA regarding parallel implementation were highlighted and several optimization techniques have been investigated. As key point to high performing implementation, a detailed memory access concept is necessary to profit from the high floating point performance of the GPUs.

Almost perfect scalability up to four GPUs was presented using the asynchronous file transfer feature to hide all inter-GPU communication behind the curl kernel execution which does not depend on the data.

Further work will include scalability tests for up to 8 GPUs, as well as hybrid CPU/GPU implementations and porting to upcoming architectures like GT300 and new compute abstractions like OpenCL.

ACKNOWLEDGMENTS

T. Warburton is partly supported by AFOSR FA9550-05-1-0473, NSF CNS-0514002 and NSF DMS-0512673. N. Gödel is partly supported by DFG travel grant CL 143/8-1.

The authors would like to thank Andreas Klöckner and Jeff Bridge for discussions and support related to DG implementation on GPUs.

REFERENCES

[1] Nvidia Corporation. "NVIDIA CUDA 2.2 Compute Unified Device Architecture Programming Guide", USA, 2009.

[2] Advanced Micro Devices, Inc., "ATI Stream Computing", Sunnyvale, USA, 2009.

[3] KHRONOS GROUP, "The OpenCL Specification Version 1.0", 2008.

[4] S. Krakiwsky, L. Turner, and M. Okoniewski, "Acceleration of Finite-Difference Time-Domain (FDTD) Using Graphics Processor Units(GPU)", *IEEE MTT-S International Microwave Symposium*, pp. 1033-1036, 2004.

[5] M. J. Inman, A. Z. Elsherbeni, J. G. Maloney and B. N. Baker, "Practical Implementation of a CPML Absorbing Boundary for GPU Accelerated FDTD Technique", *ACES Journal*, vol. 23, 2008.

[6] A. Kloeckner, T.Warburton, J. Bridge, and J. Hesthaven, "Nodal discontinuous Galerkin methods on graphics processors," *Journal of Computational Physics,* vol. 228, pp. 7863 – 7882, 2009.

[7] N. Gödel, T. Warburton, M. Clemens, "GPU Accelerated Discontinuous Galerkin FEM for Electromagnetic Radio Frequency Problems", *IEEE APS Conference Charleston,* 2009.

[8] W. Reed and T. Hill, "Triangular mesh methods for the neutron transport equation," *Los Alamos Scientific Laboratory, vol. Tech. Report*, no. LAUR-73-479, 1973.

[9] B. Cockburn, G. Karniadakis, and C.-W. Shu, "Discontinuous Galerkin Methods: Theory, Computation and Applications", *Springer,* 2000.

[10] G. Cohen, X. Ferrieres and S. Pernet, "A spatial high-order hexahedral discontinuous Galerkin method to solve Maxwell's equations in time domain", *Journal of Computational Physics*, vol. 217, pp. 340-363, 2006.

[11] J. S. Hesthaven and T. Warburton, "Discontinuous Galerkin methods for the time-domain Maxwell's equations," *ACES Journal,* vol. 19, pp. 10–29, 2004.

[12] J. S. Hesthaven and T. Warburton, Nodal Discontinuous Galerkin Methods. Springer, 2008.

[13] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs." Conference on Parallel Processing, pp. 113-122, 1995.

Nico Gödel, born 1978 in Minden, Germany, received his diploma in Electrical Engineering from the Helmut-Schmidt University, University of the Federal Armed Forces in Hamburg in 2006. From 2007 till 2010, he worked as a Research Engineer at the Chair for Theory in Electrical Engineering and Computational Electromagnetics at the Helmut-Schmidt-University, University of the Federal Armed Forces Hamburg.

Tim Warburton received a PhD in Applied Mathematics from Brown University in 1999. He is currently an Associate Professor of Computational and Applied Mathematics, at Rice University, Houston, Tx.
He co-authored the first major text on discontinuous Galerkin methods, published by Springer in 2008.

Markus Clemens, born 1968 in Wittlich, received his diploma in Mathematical Engineering ("Diplom Technomathematik") with a minor in Mechanical Engineering from the University of Kaiserslautern in 1995. In 1998 he finished his Phd at the Institute for Theory of Electromagnetic Fields at the Technische Universität Darmstadt in the field of Computational Electromagnetics. Working as postdoc at the same institute he became team leader of an interdisziplinary team of phd and postdoc researchers. In December 2003 he received his venia legendi in "Electromagnetic Theory" and "Scientific Computing". From 2004 to 2009 he was working as head of the Chair for Theory in Electrical Engineering and Computational Electromagnetics at the Helmut-Schmidt University, University of the Federal Armed Forces Hamburg. In October 2009 he took on the position as head of the Chair of Electromagnetic Theory at the Bergische Universität Wuppertal, Germany. His teaching activities involve courses in Electromagnetic Theory, Advanced Engineering Mathematics, and Computational Electromagnetics. His research activities are in the field of Computational Engineering and Mathematical Engineering. His research specifically involves the development and application of numerical simulation methods for Computational Electromagnetics and Computational Multiphysics.