

# Multi-GPU Acceleration of Algebraic Multigrid Preconditioners

Christian Richter<sup>1</sup>, Sebastian Schöps<sup>2</sup>, and Markus Clemens<sup>1</sup>

**Abstract** A multi-GPU implementation of Krylov subspace methods with an algebraic multigrid preconditioners is proposed. With this, large linear system are solved which result from electrostatic field problems after discretization with the Finite Element Method. As data is distributed across multiple GPUs the resulting impact on memory and execution time are discussed for a given problem solved with either first or second order ansatz functions.

## 1 Introduction

The solution of partial differential equations as they occur e.g. in electrostatics is of high importance in design and evaluation of virtual prototypes, e.g. of electric high-voltage system components. For this Finite Elements (FE) are very popular in electromagnetics, in particular for static and low frequency field simulations. After applying space and time discretization and possibly a nonlinear solver as e.g. the Newton-Raphson scheme, the resulting problem is a large symmetric positive definite linear algebraic system of equations. For solving these linear systems Krylov subspace method like conjugate gradients (CG) are a common approach [1]. The acceleration of the solution procedure with sophisticated preconditioners, i.e., the algebraic multigrid (AMG) method based on smoothed aggregation with graphic processing units is discussed in this paper.

As multicore systems are standard today, recent research focuses on GPUs as hardware accelerators. Sparse matrix vector (SpMV) operations can be implemented efficiently on GPUs in general [2] and particularly in applications from electromagnetics [3]. The advantages of GPU acceleration have been demonstrated for Finite

---

Bergische Universität Wuppertal, Chair of Electromagnetic Theory, 42119 Wuppertal, Germany [christian.richter@uni-wuppertal.de](mailto:christian.richter@uni-wuppertal.de), [clemens@uni-wuppertal.de](mailto:clemens@uni-wuppertal.de) · Technische Universität Darmstadt, Graduate School of Computational Engineering Institut für Theorie Elektromagnetischer Felder, 64285 Darmstadt, Germany [schoeps@gsc.ce.tu-darmstadt.de](mailto:schoeps@gsc.ce.tu-darmstadt.de)

Differences [4] as well as FE [5], [6]. The major bottleneck of a GPU is its limited local memory that determines the maximum size of a problem that can be solved at once without swapping data between the GPU and the host memory which usually has a serious impact on the performance. Consequently, when it comes to problems exceeding the memory size of a single GPU the use of multiple GPUs becomes mandatory. But even for smaller problems the CG method can be accelerated by using multiple GPUs [7].

The results in this paper extend those reported in [8]. While previously the proposed add-on of a multi-GPU-AMG solver to the CUSP library was presented for the first time, in this paper the results for second order ansatz functions and larger problems, exceeding the memory of one GPU, are discussed. Second order ansatz functions change the density of the matrix by increasing the number of non-zero matrix entries per degree of freedom. With the larger discrete problem exceeding one GPU's memory it is shown that the code can solve large problems not only in theory, but in a real-world example.

The paper is structured as follows: first the problem formulation is introduced. In Section 2 the multi-GPU implementation is described in detail. A numerical example shows the effects when taking into account multiple GPUs for solving electromagnetic problems with either first or second order ansatz functions. In the end the work is summarized.

### 1.1 Problem Formulation

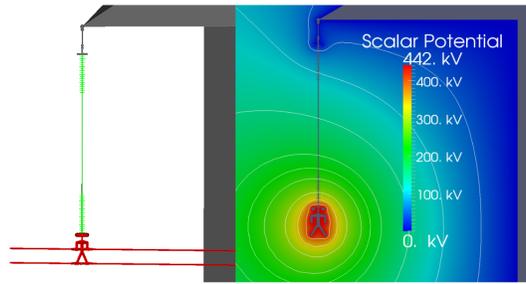
When solving an electrostatic problem, an elliptic boundary value problem has to be solved on a computational domain  $\Omega$ , i.e.,

$$-\nabla \cdot (\varepsilon(\mathbf{r})\nabla\phi(\mathbf{r})) = f(\mathbf{r}) \quad (1)$$

for  $\mathbf{r} \in \Omega$  and where  $\varepsilon$  is the spatially distributed permittivity,  $f$  a given field source,  $\phi$  the electric scalar potential with adequate boundary conditions, like a homogeneous Dirichlet constraint  $\phi|_{\partial\Omega} = 0$ . Discretizing the problem with FE results in a linear system of equations with a positive definite system matrix. The in-house simulation code 'MEQSICO' [10] is used that is capable of solving static and quasi-static electric and magnetic field problems and coupled multiphysical problems with high-order FEM ansatz functions [11]. An exemplary electrostatic problem is shown in Fig. 1 which is described by (1).

### 1.2 Algebraic Multigrid

The AMG method is used as a preconditioner for a conjugate gradient solver [12]. Apart from classical AMG it can be based on smoothed aggregation [13] as em-



**Fig. 1** CAD-Model and scalar potential of a high-voltage-insulator presented in [9]

ployed in this paper. AMG consists of two parts: in the *setup-phase*, levels of increasing coarseness are assembled from the degrees of freedom. To enable the different grid levels to interact the prolongation and restriction operators are constructed, which connect two consecutive levels. With the so-called Galerkin product, a triple matrix product, the system matrix of the coarser level is constructed.

The multigrid preconditioner is applied in every iteration step within the *solve-phase* of the CG method. At first the given linear system is subject to a smoothing step and the afterwards computed residual is restricted to the next coarse level. Within this next level the described function is called recursively. After returning from the next coarse level, the result is corrected by an error calculated on the coarser grid level and a smoothing step is applied again. Instead of calling further recursive calls the system is solve on the coarsest level. The coarsest system is solved by direct or iterative solvers. Details of this V-cycle approach are given in [14, 15].

While the complex calculations of the setup phase are performed only once for a given system matrix, the solve phase is executed in every iteration step. Consisting only of some SpMVs the multigrid V-cycle in the solve phase is less time intense. These operations can be performed very efficiently on GPUs [2].

## 2 Algebraic Multigrid on Multiple GPUs

The CUSP library [16] is a well-established and fast library for solving linear equation systems on GPUs. Providing efficient implementations for matrix-vector operations and a set of solvers including an AMG-Preconditioner [17] it is well suited as a starting point for our multi-GPU approach. The AMG preconditioner [17] has a high memory demand due to the multiple grids, each storing its own system matrix as well as matrices for restriction and prolongation. To overcome the limitations we propose to distribute the data across multiple GPUs. As CUSP is an evolving software library the decision was made to implement a multi-GPU extension as an add-on to this library. The environment uses templated C++ classes and can interact with CUSP. Main parts of the add-on are classes for multi-GPU vectors and matrices, communication routines for data exchange between the GPUs and a multi-GPU

PCG solver with an AMG-preconditioner that solves the system on multiple GPUs in parallel.

## 2.1 Multi-GPU Datatypes

The major part of memory is spent on the storage of matrices. Therefore redundancy has to be avoided. To achieve this the matrices are split up in a row wise manner and the resulting parts are copied to the individual GPUs. During the splitting process the input matrix is converted into the Compressed Sparse Row (CSR) format, split up and the resulting parts are reconverted. Due to this the splitup is performed fast with almost no calculation effort and the resulting parts are load balanced as the entries per row remain unchanged supposing that each row has approximately the same number of non-zero elements. With respect to the construction of the multi-GPU matrix class and due to the fact that vectors have only low memory demand compared to a matrix, the vector class holds a full copy of every vector on each GPU. The vector part corresponding to the GPU is defined on the full vector via a vector view, i.e., a kind of pointer. When performing an operation it is executed simultaneously on all GPUs using OpenMP. During a vector-vector operation each GPU performs the operation only on the corresponding vector parts. A matrix-vector operation is performed on every GPU using the CUSP SpMV routine with the full vector as input and the corresponding vector view as output. Therefore it is important that the whole input vector is up to date on each GPU. This has to be ensured by communication routines.

## 2.2 Inter-GPU Communication

The exchange of data between the GPUs is the most critical part of the implementation. Sparse operations on a single GPU are already limited by the bandwidth of the connection between the GPUs global memory and its processing unit. When it comes to inter-GPU communication there is a large gap between the GPUs internal bandwidth and the connecting Peripheral Component Interconnect Express (PCIe) bandwidth. A contemporary GPU like the Nvidia Tesla K20X as used in this work has a theoretical bandwidth of 250 GB/s. The PCIe bus which connects the GPUs has a bandwidth of 8 GB/s. Therefore sophisticated communication schemes have been developed to minimize the burden of communication. With the *copy-In* routine a whole vector is distributed from one GPU to all GPUs involed in the calculation. With "direct access"<sup>1</sup> data can be copied from one GPU to another without going through the host. Due to this and by using asynchronous memory copy functions data can be copied between different GPUs in parallel. With these measures the

---

<sup>1</sup> this technology was introduced with CUDA 4.0 within the framework of a "unified address space", i.e., a virtual address space for the host and all GPUs attached

bandwidth scales linearly with the number of parallel processes. As a result the vector is not copied to each GPU one after another but instead it is realized as follows: the first part of the vector is copied from the first GPU to the second one, from the second to the third and so on. In the meantime the second memory segment is transferred from the first to the second GPU in parallel. This can be realized because of the two copy engines of contemporary GPUs enabling them to send and receive data at the same time. The most important routine is the *gather-nn* routine. It is used when each GPU holds only its own piece of data and a vector has to be updated such that every vector holds a up-to-date version of the whole vector. This is the case before every SpMV. Here the same principle is used as described above. Each GPU copies its piece data to the next GPU in the cycle. When the transmission is finished the GPU sends the next piece of data it has just received to the next GPU. In this way each GPU sends and receives data during the whole procedure maximizing the parallel throughput.

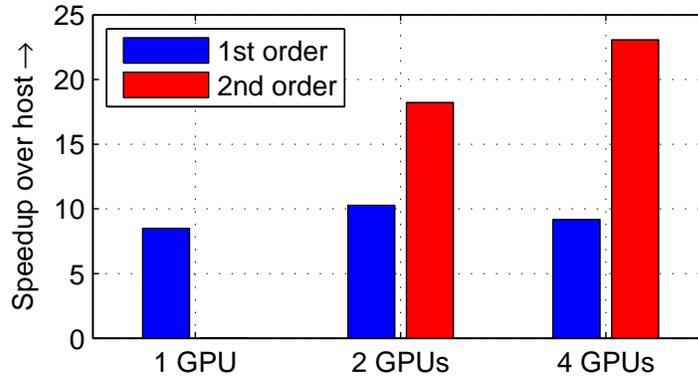
### 2.3 Preconditioned Conjugate Gradients

The AMG preconditioner is set up by the CUSP library. As shown in [18] it should be set up on the host to overcome memory limitations of a single GPU. Within the multi-GPU splitup routine, the preconditioner is divided level by level and distributed across all GPUs involved. The multi-GPU CG routine then solves the problem by handing over the multi-GPU preconditioner and the original right hand side and solution vectors. The routine is build analogously to the CUSP AMG-CG, but uses multi-GPU routines. Due to the implementation only minimal changes are necessary in an existing code and the behavior in terms of residual reduction per step is almost identical.

## 3 Numerical Example

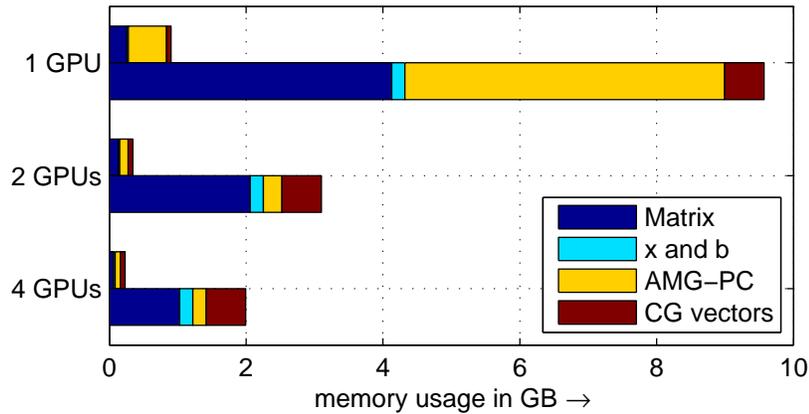
As an example a real-world FE problem is solved using first and second order ansatz functions. The example, a high voltage insulator as is presented in [9], is shown in Fig. 1. The discrete model has  $1.5 \times 10^6$  degrees of freedom and a linear system matrix consisting of  $21 \times 10^6$  nonzero entries. When using second order ansatz functions the linear problem expands to  $12 \times 10^6$  dof and  $340 \times 10^6$  nonzero matrix entries. In both cases the problem is solved to a relative residual norm of  $1 \times 10^{-12}$ . Calculations are performed on a server running CentOS 6.5. It is equipped with two Intel Xeon E5-2670 CPUs and 128 GB RAM. Four Nvidia Tesla K20Xm GPUs are attached to the host. To ensure data integrity, error-correcting code (ECC) is enabled. Thus the effective bandwidth of each GPU is reduced from 250 GB/s to 200 GB/s. Host parallelization is done by OpenMP on the host. On the devices architecture model 3.5 is used. The code is compiled with CUDA 5.0, Thrust 1.8.0, CUSP

0.4.0 and GCC 4.4.7 with -O3. For comparison the problem is also solved on the



**Fig. 2** Speedup of the Solve-Phase for the First- and Second-Order-Problem on a varying number of GPUs

host using the CUSP host version which has been shown to outperform [18] state of the art libraries like Petsc [19] or Trilinos ML [20]. The setup phase is performed on the host where it is stored and distributed to the GPUs. The preconditioner is only setup once and can be reused for multiple right hand sides, i.e., for several timesteps in a quasistatic simulation. The speedup of the GPU implementations over the host implementation is depicted in Fig. 2. It shows the individual speedup when solving the problem with first and second order ansatz functions for a varying number of GPUs. The problem cannot be solved on one GPU with second order ansatz functions due to memory limitations and therefore no results are presented. One can see that with first order ansatz functions a speedup of 7.7 times is achieved when using one GPU. It can be increased to a factor 10.8 with two GPUs but decreases to 9.5 when using four GPUs. This has two reasons: firstly the communication processes become so costly that the speedup in calculation cannot compensate for them. The data per GPU is not sufficient to keep the GPUs busy. Secondly every calculation and data movement operation needs a certain fixed time to be launched. This has a higher effect when the time to perform the operation is lower. The second order case differs from the first order one. The speedup is again more than doubled compared to the first order case. It is increased to 18.1 on two GPUs and over 23 when using four GPUs. This speedup over the first order GPU calculation has two major reasons. At first the time the GPUs spend for calculations is larger because of the increased work each GPU has to do. This minimizes launch effects. Then the matrix itself is much denser with second order ansatz functions with an average of over 28.2 instead of 14.7 entries per row of the system matrix. This means that the calculations increase compared to the number of degrees of freedom, which are transferred at every individual communicational operation.



**Fig. 3** Memory-Usage per GPU for the First- (upper bar) and Second-Order-Problem (lower bar) on a varying number of GPUs

Fig. 3 shows the memory consumption for the given problem. It is separated by the order of the ansatz functions and number of GPUs in use. Each bar is the sum of the individual parts of the CG-solver. The use of second order ansatz functions is shown to lead to a much higher memory demand. With the number of GPUs involved the memory demand per GPU of the matrices decreases linearly because they are split up and no information is saved redundantly. In contrast to this the memory demand for a vector remains unchanged because each GPU has to hold the full vector. But as the matrices memory demand is dominating the overall scaling remains almost linear. Another reduction of memory demand can be achieved by erasing redundancies between the matrix and the AMG preconditioner. In CUSP the system matrix is saved redundantly in the preconditioner as the system matrix for the finest level. As this is not needed in the proposed add-on a further reduction is obtained.

## 4 Conclusion

The limitations of a single GPU can be overcome by using multiple GPUs for solving high dimensional discrete electric or magnetic field problems. This has been shown with an add-on to the CUSP library that enables multi-GPU computing for FE simulations with first and second order ansatz functions. Memory consumption scales approximately linear with the number of used GPUs. Furthermore, significant speedups can be achieved by multiple GPUs, even though inter-GPU communication has to be taken into account. Especially higher order simulations can be accelerated significantly due to the higher density of their system matrices.

## References

1. Y. Saad, *Iterative methods for sparse linear systems*, 2nd ed. Boston: SIAM, 2003.
2. N. Bell and M. Garland, “Efficient sparse matrix-vector multiplication on CUDA,” NVIDIA Corporation, NVIDIA Technical Report NVR-2008-004, 2008.
3. M. Mehri Dehnavi, D. M. Fernández, and D. Giannacopoulos, “Finite-Element sparse matrix vector multiplication on graphic processing units,” *IEEE Trans. Magn.*, vol. 46, no. 8, pp. 2982–2985, 2010.
4. C. Richter, S. Schöps, and M. Clemens, “GPU acceleration of finite differences in coupled electromagnetic/thermal simulations,” *IEEE Trans. Magn.*, vol. 49, no. 5, pp. 1649–1652, 2013.
5. M. Mehri Dehnavi, D. M. Fernández, and D. Giannacopoulos, “Enhancing the performance of conjugate gradient solvers on graphic processing units,” *IEEE Trans. Magn.*, vol. 47, no. 5, pp. 1162–1165, 2011.
6. M. Mehri Dehnavi, D. M. Fernández, and J.-L. Gaudiot, “Parallel sparse approximate inverse preconditioning on graphic processing units,” *IEEE Trans. Par. Dist. Sys.*, vol. 24, no. 9, pp. 1852–1862, 2013.
7. M. Verschuur and A. C. Jalba, “Analysis and performance estimation of the conjugate gradient method on multiple GPUs,” *Parallel Computing*, vol. 38, no. 1011, pp. 552 – 575, 2012.
8. C. Richter, S. Schöps, and M. Clemens, “Multi-GPU acceleration of algebraic multigrid preconditioners for elliptic field problems,” *IEEE Trans. Magn.*, accepted.
9. H. Ye, M. Clemens, and J. Seifert, “Electro-quasistatic field simulation for the layout optimization of outdoor insulation using microvaristor material,” *IEEE Trans. Magn.*, vol. 49, no. 5, pp. 1709 – 1712, 2013.
10. T. Steinmetz, M. Helias, G. Wimmer et al. “Electro-quasistatic field simulations based on a discrete electromagnetism formulation,” *IEEE Trans. Magn.*, vol. 42, no. 4, pp. 755–758, 2006.
11. P. Monk, *Finite Element Methods for Maxwell’s Equations*. Oxford University Press, 2003.
12. K. Stüben, “Algebraic multigrid (AMG): An introduction with applications,” GMD, Report 53, 1999.
13. P. Vanek, J. Mandel and M. Bresina, “Algebraic Multigrid By Smoothed Aggregation For Second and Fourth Order Elliptic Problems,” *COMPUTING*, , vol. 56, pp. 179–196, 1996
14. Y. Shapira, *Matrix-based multigrid: theory and applications*, ser. Numerical methods and algorithms. Springer, 2008.
15. U. Trottenberg, C. Oosterlee and A. Schuller, *Multigrid*. Academic Press, Inc., 2001.
16. N. Bell and M. Garland, “CUSP: Generic parallel algorithms for sparse matrix and graph computations,” 2012, version 0.4.0.
17. N. Bell, S. Dalton, and L. N. Olson, “Exposing fine-grained parallelism in algebraic multigrid methods,” *SIAM J. Sci. Comput.*, vol. 34, no. 4, 2012.
18. C. Richter, S. Schöps, and M. Clemens, “GPU acceleration of algebraic multigrid preconditioners for discrete elliptic field problems,” *IEEE Trans. Magn.*, vol. 50, no. 2, pp. 461–464, 2014.
19. S. Balay, J. Brown, K. Buschelman, et al., “PETSc users manual,” Argonne National Laboratory, Tech. Rep. ANL-95/11 - Rev. 3.4, 2013.
20. M. Gee, C. Siefert, et al., “ML 5.0 smoothed aggregation user’s guide,” Sandia National Laboratories, Tech. Rep. SAND2006-2649, 2006.