Single and Dual-GPU Generalized Sparse Eigenvalue Solvers for Finding a Few Low-Order Resonances of a Microwave Cavity Using the Finite-Element Method

Adam DZIEKONSKI, Michal MROZOWSKI

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

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

Submitted August 30, 2018 / Accepted September 3, 2018

Abstract. *This paper presents two fast generalized eigenvalue solvers for sparse symmetric matrices that arise when electromagnetic cavity resonances are investigated using the higher-order finite element method (FEM). To find a few low-order resonances, the locally optimal block preconditioned conjugate gradient (LOBPCG) algorithm with null-space deflation is applied. The computations are expedited by using one or two graphical processing units (GPUs) as accelerators. The performance of the solver is tested for single and dual GPU hardware setups, making use of two types of GPU: NVIDIA Kepler K40s and NVIDIA Pascal P100s. The speed of the GPU-accelerated solvers is compared to a multithreaded implementation of the same algorithm using a multicore central processing unit (CPU, Intel Xeon E5-2680 v3 with twelve cores). It was found that, even for the least efficient setups, the GPU-accelerated code is approximately twice as fast as a parallel CPU-only implementation.*

Keywords

FEM, generalized eigenvalue problem, GPU, Maxwell's equations, resonators

1. Introduction

Computational electromagnetics (CEM) concerns the design and implementation of fast and accurate solvers of Maxwell's equations. The ultimate goal is to solve complex problems in the shortest possible time. To achieve this goal, algorithms have to be considered in the context of the hardware they will be implemented on. As hardware evolves, known numerical techniques need to be reexamined and modified in order to take advantage of the possibilities offered by new hardware. GPU-computing is an emerging trend in computational electromagnetics. Initially, GPU-acceleration was considered for the finite-difference time-domain (FDTD) method [1–4]. In recent years, some effort has also been devoted to taking advantage of the processing power of graphics accelerators for other techniques, such as the method of moments [5–7], the multilevel fast multipole (MFML) algorithm [8], and the discontinuous Galekin (DG) method [9], as well as other techniques [10]. In this paper, we consider GPU-acceleration of the finite element Method—one of the most powerful and versatile numerical techniques, which is often applied to the solution of boundary value problems that arise in electromagnetics. To date, most publications on FEM in the context of electromagnets have been related to the solution of a linear system of equations [11–13] and FEM matrix generation and assembly [13–15]. In this paper, we concentrate on finding the solution of the generalized eigenvalue problems that emerge when FEM is applied to simulate the free oscillation of microwave cavities. GPU-acceleration of eigenvalue solvers is less frequently disused in the literature, and in most cases standard, rather than generalized, eigenvalue problems are considered [16–19]. Solvers for standard eigenvalue problems are not suitable for FEM analysis, so we focus on generalized eigenvalue problems, extending our recent work on this topic [20].

2. Finite Element Method

Let us consider a lossless dielectric-loaded electromagnetic cavity Ω enclosed by a perfect electric conductor boundary S. The behavior of the electric field inside Ω and the frequencies of free oscillations inside are determined by the vector Helmholtz equation:

$$\nabla \times \nabla \times \vec{E} - k_0^2 \epsilon_r \vec{E} = 0 \tag{1}$$

where $\vec{E}$ is the electric field, $k_0 = \omega/c$ is the wavenumber, ϵ_r is the relative permittivity, ω is the angular frequency, and c is the speed of light in vacuum. The nonzero frequencies for which the above equation has a nontrivial solution determine the resonances of the cavity.

To solve the vector Helmholtz equation in 3D, we use the finite-element method with a tetrahedral mesh and higher-order basis functions [21], [22]. To this end, we work with the weak formulation of (1):

$$\int_{\Omega} \left(\nabla \times \vec{w} \cdot \nabla \times \vec{E} - k_0^2 \vec{w} \cdot \epsilon_{\mathrm{r}} \vec{E} \right) \mathrm{d}\Omega = 0 \tag{2}$$

where $\vec{w}$ is a vector testing function. We then use the Galerkin method with hierarchical vector basis functions up to the third order [23], arriving at:

$$(\mathbb{K} - k_0^2 \mathbb{M})\, \mathbf{e} = 0 \tag{3}$$

where $\mathbb{K}, \mathbb{M} \in \mathbb{R}^{n \times n}$ are large and sparse real-valued stiffness and mass matrices, respectively, $\mathbf{e} \in \mathbb{R}^n$ is a vector of expansion coefficients for the basis functions, and n is the number of degrees of freedom (DoF). Additionally, the matrix $\mathbb{M}$ is positive definite. Equation (3) defines the generalized eigenvalue problems. Since matrices are real and symmetric, and one of them is positive definite, the eigenvalues are real and nonnegative.

3. Finding Small Nonzero Eigenvalues

The system matrix that emerges from FEM using higher-order basis functions is large and sparse. This means that direct solution methods for eigenproblems cannot be applied. The only way to find the resonances and modal fields is to use iterative techniques. In practice, we are interested in a few low-order resonances that correspond to eigenvalues with small magnitudes. One algorithm recommended for finding a few algebraically smallest eigenvalues of symmetric positive definite matrix pencils is the locally optimal block preconditioned conjugate gradient (LOBPCG) method proposed by Knyazev [24]. With a good preconditioner, this method converges rapidly. Moreover, it does not require any matrix factorization and, because it involves a three-term recurrence, its memory requirements are relatively low. Finally, the basic operation in the algorithm is a sparse matrix–vector product. These features are desirable from the point of view of efficiently using the massive parallelization capabilities offered by state-of-the-art graphics processing units. The LOBPCG algorithm cannot be applied directly to the FEM problems resulting from the weak form (2), as the $\nabla \times \nabla \times (\cdot)$ operator yields zero when applied to the gradient of any scalar function. As a result, the eigenproblem (3) has multiple zero eigenvalues. Such eigenvalues would be found by LOBPCG, but they are of no interest, since they are not physical solutions fulfilling Maxwell's equations. More precisely, the model field associated with each zero eigenvalue does not fulfill the condition that $\nabla \cdot \epsilon_{\mathrm{t}} \vec{E} = 0$. To get rid of these spurious solutions, a zero divergence condition can be imposed on the vectors produced during each iteration of LOBPCG [25]. This process can be called null-space filtering, as imposing the divergence-free condition is, in practice, implemented as in a deflation using a projector that ensures that the projected vectors are $\mathbb{M}$-orthogonal to the null-space. The projected vectors thus have no components from the null-space, and are then used to construct model fields (eigenvectors) that correspond to nonzero eigenvalues and fulfill $\nabla \cdot \epsilon_{\mathrm{t}} \vec{E} = 0$.

The LOBPCG method with null-space filtration is shown in Algorithm 1. The null-space filtering is applied in lines 2 and 13. In practice, this is implemented as a solution of a relatively small system of sparse equations with multiple right-hand sides. The system matrix for null-space filtration is identical to the system matrix obtained when solving the Poisson equation with the Lagrange finite elements. The matrix is given by $\mathbf{Y}^{\mathrm{T}} \mathbb{M} \mathbf{Y}$), with $\mathbf{Y}$ being a discrete gradient operator with only two nonzero elements per row (1 or -1) [25]. To solve the Poisson equation, we use the conjugate gradient method with a hierarchical multilevel preconditioner operating in a V-cycle (Algorithm 2 [22]. The hierarchical multilevel preconditioner (Hie-ML) is shown in lines P.1–P.11 of Algorithm 2. The preconditioner takes advantage of the hierarchy of the basis functions used in FEM. In our case, the basis functions up to the third order are considered (QTCuN) [23], so we used three levels in a V-cycle. On the lowest level (P.4), the direct solution of a linear system is needed. However, since on this level only the first-order basis functions are involved, the system is very small and its solution does not pose any problem, For the smoothing that is performed when transfers between levels occur during restriction and prolongation (lines P.6–P.10), we used a few weighted Jacobi iterations.

Another important element is the preconditioner $\mathscr{P}$ used in the LOBPCG (Algorithm 1, line 12). In this paper, we employ for this a sequence of operations approximately equal to the inverse of the matrix $\mathbb{A} = \mathbb{K} - \kappa \mathbb{M}$, on a vector or a block of vectors (κ is a real scalar chosen to be smaller than, but close to, the smallest nonzero eigenvalue of the matrix pencil $(\mathbb{K}, \mathbb{M})$. This is implemented as a few iterations of the preconditioned conjugate gradient method (PCG-V)—the same method used for null-space filtering. In other words, in Step 12 of the LOPCG algorithm, instead of computing $\mathbf{H}_k = \mathscr{P}^{-1} \mathbf{R}_k$, we used an approximate solution of the matrix system $(\mathbb{K} - \kappa \mathbb{M}) \mathbf{H}_k = \mathbf{R}_k$.

Note that, except for the direct solution on the lowest level (Algorithm 2, line P.4), almost all other steps in the LOBPCG algorithm can be expressed in terms of a matrix-times-vector operation or simple BLAS1-type operations. Also, the preconditioned conjugate-gradient algorithm involves a sparse matrix–vector product (SpMV). This is typical of all iterative techniques based on Krylov-spaces. In order to execute the iterations rapidly, the performance of the sparse matrix–times vector product should be increased. SpMV multiplication can be executed much more rapidly on GPUs than on multicore Central Processing Units (CPUs).

Algorithm 1 Stabilized LOBPCG for real symmetric problems. Inputs: $\mathbb{K}$ and $\mathbb{M}$ are sparse real-valued symmetric $n \times n$ matrices, ϵ is the assumed accuracy, $\mathscr{P}$ is a preconditioner, $\mathbf{Y}$ is a basis in the nullspace, $\mathbf{X}_0$ is the initial block vector of size $n \times (q+1)$, q is the number of eigenvalues to be computed, and MaxIter is the maximum number of iterations. γ is the shift value used in the sorting algorithm. The outputs of the LOBPCG method are the q smallest nonzero eigenvalues $\{\sigma_1, \ldots, \sigma_q\}$, stored in a diagonal matrix Σ_{output}, and a dense block vector $\mathbf{X}_{\text{output}}$ of size $n \times q$ consisting of the q respective eigenvectors. In this algorithm, σ is the eigenvalue corresponding to k_0^2 from (3).

1: Initialize $\tilde{\mathbf{P}}_0 = \mathbf{P}_0 = []$
2: $\mathbf{X}_0 \leftarrow \mathbf{X}_0 - \mathbf{Y}\,(\mathbf{Y}^{\mathrm{T}}\mathbb{M}\mathbf{Y})^{-1}((\mathbb{M}\mathbf{Y})^{\mathrm{T}}\mathbf{X}_0)$ ▷ Make $\mathbf{X}_0$ $\mathbb{M}$-orthogonal to the nullspace
3: $\mathbb{M}$-orthogonalize columns of $\mathbf{X}_0$
4: Compute $(\mathbf{X}_0^{\mathrm{T}}\mathbb{K}\mathbf{X}_0)\tilde{\mathbf{S}}_0 = \tilde{\mathbf{S}}_0\tilde{\Sigma}_0$, where $\tilde{\Sigma}_0 = \mathrm{diag}(\sigma_1, \ldots, \sigma_{q+1})$ ▷ Spectral decomposition
5: $(\Sigma_0, \mathbf{S}_0) \leftarrow$ Eigenpairs $(\sigma_i, \tilde{\mathbf{S}}_0\mathbf{e}_i)$ sorted by $|\sigma_i - \gamma|$ in ascending order.
6: $\mathbf{X}_0 \leftarrow \mathbf{X}_0\mathbf{S}_0$
7: **for** $k = 0 : (\mathrm{MaxIter} - 1)$ **do**
8: Compute residuals $\tilde{\mathbf{R}}_k = \mathbb{K}\mathbf{X}_k - \mathbb{M}\mathbf{X}_k\Sigma_k$
9: Find $D = \left\{i : \|\tilde{\mathbf{R}}_k\mathbf{e}_i\|_2 > \epsilon\right\}$, $\tilde{q}$ = size of D
10: **if** $(\tilde{q} = 1$ **AND** i=q+1) **OR** $(\tilde{q} = 0)$ **then** ▷ Convergence check
exit
end if
11: Let $\mathbf{R}_k = [\tilde{\mathbf{R}}_k(:, j)]_{j \in D}$
12: Apply preconditioner $\mathbf{H}_k = \mathscr{P}^{-1}\mathbf{R}_k$
13: $\mathbf{H}_k \leftarrow (\mathbf{I} - \mathbf{Y}(\mathbf{Y}^{\mathrm{T}}\mathbb{M}\mathbf{Y})^{-1}((\mathbb{M}\mathbf{Y})^{\mathrm{T}})\mathbf{H}_k$ ▷ Filtering out nullspace components
14: $\mathbf{H}_k \leftarrow \mathbf{H}_k - \mathbf{X}_k((\mathbb{M}\mathbf{X}_k)^{\mathrm{T}}\mathbf{H}_k)$▷ Orthogonalization vs. eigenvector approximations
15: If $\tilde{\mathbf{P}}_k$ nonempty, set $\mathbf{P}_k = [\tilde{\mathbf{P}}_k(:, j)]_{j \in D}$
16: Compute $\tilde{\mathbf{K}} = [\mathbf{X}_k, \mathbf{H}_k, \mathbf{P}_k]^{\mathrm{T}}\mathbb{K}[\mathbf{X}_k, \mathbf{H}_k, \mathbf{P}_k]$, $\tilde{\mathbf{M}} = [\mathbf{X}_k, \mathbf{H}_k, \mathbf{P}_k]^{\mathrm{T}}\mathbb{M}[\mathbf{X}_k, \mathbf{H}_k, \mathbf{P}_k]$
17: Compute $\tilde{\mathbf{K}}\tilde{\mathbf{S}}_k = \tilde{\mathbf{M}}\tilde{\mathbf{S}}_k\tilde{\Sigma}_k$, where $\tilde{\Sigma}_k = \mathrm{diag}(\sigma_1, \ldots, \sigma_{(q+1)+2\tilde{q}})$ ▷ Spectral decomposition
18: $(\Sigma_k, \tilde{\mathbf{S}}_k) \leftarrow$ Eigenpairs $(\sigma_i, \tilde{\mathbf{S}}_k\mathbf{e}_i)$ sorted by $|\sigma_i - \gamma|$ in ascending order.
19: $\mathbf{S}_k = \tilde{\mathbf{S}_k}[\mathbf{e}_1, \ldots, \mathbf{e}_{q+1}]$, $\Sigma = \Sigma_k(1 : q+1, 1 : q+1)$
20: $\tilde{\mathbf{P}}_k = [\mathbf{R}_k\mathbf{P}_k]\mathbf{S}_k(q+1 : (q+1+2\tilde{q}), :)$
21: $\mathbf{X}_k \leftarrow \mathbf{X}_k\mathbf{S}_k(1 : q+1, :) + \tilde{\mathbf{P}}_k$
22: **end for**
23: Eigenpairs: $\mathbf{X}_{\text{output}} = \mathbf{X}(1 : n, 1 : q)$ and $\Sigma_{\text{output}} = \Sigma(1 : q, 1 : q)$.

Algorithm 2 Preconditioned conjugate gradient method (PCG-V) with hierarchical multilevel preconditioner (*Hie-ML*) operating with a single V-cycle. The algorithm solves the system of equations $\mathbb{A}\mathbf{x} = \mathbf{b}$. Matrices $\mathbb{A}_{i,j}$, $i, j = 1, 2, 3$ are blocks of $\mathbb{A}$ corresponding to various orders of basis and testing functions.

1: $\mathbf{r} = \mathbf{b}$ - $\mathbb{A}\mathbf{x}$
2: $\mathbf{d} = \mathscr{R}^{-1}\,\mathbf{r}$
3: $\delta_{\text{new}} = \mathbf{r}^{\mathrm{T}}\,\mathbf{d}$
4: **while** $\|\mathrm{r}\| > \epsilon_{\text{PCG-V}}$
5: $\mathbf{d} = \mathbb{A}\mathbf{d}$
6: $\alpha = \frac{\delta_{\text{new}}}{\mathbf{r}^{\mathrm{T}}\mathbf{d}}$
7: $\mathbf{x} = \mathbf{x} + \alpha\mathbf{x}$
8: $\mathbf{r} = \mathbf{r}$ - $\alpha\mathbf{q}$
9: $\mathbf{s} = \mathscr{R}^{-1}\,\mathbf{r}$
10: $\delta_{\text{new}} = \delta_{\text{old}}$
11: $\delta_{\text{new}} = \mathbf{r}^{\mathrm{T}}\,\mathbf{s}$
12: $\beta = \frac{\delta_{\text{new}}}{\delta_{\text{old}}}$
13: $\mathbf{d} = \mathbf{s} + \beta\mathbf{d}$
14: **end while**

Hierarchical preconditioner $\mathbf{z} = \mathscr{R}^{-1}\,\mathbf{r}$:

P.1: $\mathbf{z}$ = Hie-ML($\mathbf{r}$,i)
P.2: $\mathbf{z} = 0$
P.3: **if** i == 1 then
P.4: $\mathbf{z} = \mathbb{A}_{11}^{-1}\mathbf{r}$ //solve the lowest level
P.5: **else**
P.6: smoothing($\mathbf{z}, \mathbf{r}$) //the highest level
P.7: $\mathbf{r}^{i-1} = \mathbf{r} - \mathbb{A}_{i-1,i}\,\mathbf{z}$
P.8: $\mathbf{z}^{i-1}$ = Hie-ML($\mathbf{r}$,i-1)
P.9: $\mathbf{r}^{i} = \mathbf{r} - \mathbb{A}_{i,i-1}\,\mathbf{z}^{i-1}$
P.10: smoothing($\mathbf{z}, \mathbf{r}$) //the highest level
P.11: **end if**

4. Implementation

Two GPU-accelerated implementations of the LOBPCGG algorithm were developed. The first was intended for a single-GPU operation. In the single-GPU version, we applied preconditioning and null-space filtering to the GPU-accelerated PCG-V solvers developed previously [11], [12]. In the PCG-V with hierarchical multilevel preconditioner solvers, a direct solution of a sparse system of equations is needed on the lowest level of the V-cycle. This step is executed on a CPU, and we employed a shared-memory multiprocessing parallel direct sparse solver, Intel MKL PARDISO. For BLAS-like operations on a GPU, we used CUDA and cuSPARSE (v8.0). One exception was the sparse-matrix vector product (SpMV). As explained above, this operation is crucial for all iterative solvers based on Krylov spaces, and should be optimized for the hardware architecture that is used. The performance of this computational kernel depends on a sparse matrix storage format. Here we used the Sliced ELLR-T format that we designed [26] specifically for accelerating the iterative solution of large sparse real-valued and complex-valued systems of equations. In this format, the sparse matrix is first permuted according to the number of nonzero elements, and is then divided into submatrices (slices) consisting of a certain (fixed) number of adjacent rows. Next, the rows in which the number of nonzero entries is not a multiple of 16 are padded with zeros. The goal of this padding is to obtain coalesced and aligned access to global memory. When SpMV is performed, multiple threads are assigned to each row. The number of threads per operation on a particular row depends on the average number of nonzero entries in the rows of the sparse matrix. For the real-valued FEM matrices used in our code, Sliced ELLR-T performs at 36.6 GFlop/s with SpMV on the P100 GPU, while the CSR format used by the SpMV in NVIDIA's original cuSPARSE library (v. 8.0) achieves 21.2 GFlop/s for the same operation.

The second GPU-accelerated implementation uses two GPUs to perform preconditioning in Step 12 of the algorithm. For this implementation, we used our dual-GPU PCG-V solver [27]. All operations other than Step 12 were carried out on one GPU. It is crucial to the efficiency here to split the system matrix between two GPUs so that computations performed by each GPU are balanced and so data transfer between accelerators in each PCG-V iteration is minimized. It is also essential for the Hie-ML preconditioner that the matrices on each GPU can be split into a nine-block structure, with each block corresponding the order of the basis and testing function. The data distribution technique proposed in [27] first splits tetrahedra into two sets, with approximately the same number of tetrahedra in each set. Each set corresponds to one GPU. To this end, a mesh is represented as a graph and a graph partitioning algorithm is employed; this chooses the tetrahedra for the two sets in such a way that the number of common edges for the subgraphs belonging to both sets is minimal. Based on the subgraphs, the mass and stiffness matrices are generated separately for each GPU [14]. This guarantees that the block structure of these matrices preserve the hierarchy of the basis and testing functions. After this step, each GPU contains its local data plus a small number of rows that are common to both sets (typically 1%–2% of the total number of rows). The matrices are then converted to Sliced ELLR-T format. When the SpMV product is evaluated, each GPU uses its local data, also on common rows. Once this has been done, synchronization is enforced and the results of multiplication on common rows are exchanged between GPUs.

5. Numerical Results

All numerical tests were executed on an Intel Xeon (E5-2680 v3, 2.5 GHz, twelve cores) with 256 GB memory and two hardware setups with one or two GPUs. The GPU accelerators we used were either NVIDIA Tesla K40s (Kepler with 2888 CUDA cores) or P100s (Pascal with 3584 CUDA cores). Each accelerator was equipped with 12 GB GPU RAM. We also developed a reference (CPU-based) implementation based procedures from the Intel Math Kernel Library, so that the reference computation was performed in parallel using twelve CPU cores and procedures optimized for best performance on Intel multicore processors.

In order to investigate the performance of all solver implementations, the problem of finding five low-order resonances of a dielectric loaded resonator was considered. This involves a cylindrical PEC cavity loaded with a dielectric disc ($\epsilon_r = 37$) placed on a dielectric support with $\epsilon_r = 2.2$. The resonator is a lossless variant of the cavity investigated in [28]. FEM matrices were generated using the higher-order FEM code described in [29], using the dimensions provided in [28]. Table 2 shows the simulation parameters forthe LOBPCG and matrices. Note that the accuracy of the preconditioner $\epsilon_{\text{PCG-V}}$ is set to a high value of 10^{-2}. Null-space filtering was carried out by iteratively solving a small system of equations with $k = 320134$ rows until the error dropped below 10^{-6}.

Table 3 shows the eigenvalues and resonance frequencies for the first five low-order modes of the cavity calculated with LOBPCG with null-space filtering. The values found by all our solvers are identical and are in very good agreement with the resonances computed using CST software.

The times taken by the CPU-accelerated and GPU-accelerated LOBPCG implementations for various steps of the algorithm and hardware setups are shown in Tab. 1. Using GPUs as accelerators involves some overhead due to the extra time needed for memory allocation and transfer in Step 1 of the algorithm. In this step, the GPU implementations are slower than the CPU-only version. However, the time taken by pivotal phases, such as the preconditioner (Phase 12) and the null-space filtering (Phases 2 and 13) is significantly reduced in the GPU implementations. As a result, the time taken by the complete LOBPCG is reduced from 254 seconds to 143 and 108 seconds respectively for the K40 and P100 GPU accelerators. In the dual-GPU arrangement,

LOBPCG Phase	**CPU** (T_1)	**GPU** (T_2)	**GPU** (T_3)	**GPU** (T_4)	**GPU** (T_5)
$\mathscr{P}$ = PCG-V	1 × Xeon	1 × K40	2 × K40	1 × P100	2 × P100
$\mathbb{K}$,$\mathbb{M}$	1 × Xeon	1 × K40	1 × K40	1 × P100	1 × P100
1	21.4	49.2	53.4	53.1	45.5
2	4.5	1.9	1.9	1.3	1.4
12	141.9	62.7	49.8	34.7	30.2
13	54.1	21.2	21.3	12.5	12.9
16	22.9	6.4	6.4	2.8	2.9
3–11,14–15,17–22	8.1	1.7	1.7	3.0	3.0
23	0.9	0.2	0.2	0.1	0.1
1–23	253.9	143.3	134.6	107.5	96.1

Tab. 1. Time (in seconds) taken by CPU-based and GPU-based implementations of the LOBPCG algorithm for a test problem from Tab. 2.

Description	**LOSSLESS**
problem size n	1,343,373
nonzero elements n_z	110,296,368
Null-space size k	320,134
number of ev. to be found q	5
shift value (κ) in $\mathscr{P}$	8800
shift value (γ) in sort	8800
Tolerance for evs. $\epsilon_{\text{sLOBPCG}}$	10^{-4}
Tolerance for precond. $\epsilon_{\text{PCG-V}}$	10^{-2}
Tolerance in NSF ϵ_{NSF}	10^{-6}

Tab. 2. Test problem description

$\sigma = k_0^2$	**freq** [GHz]
8894.458	4.500
12831.709	5.405
12955.003	5.431
15588.509	5.957
17542.564	6.320
17651.925	6.339

Tab. 3. Eigenvalues and corresponding frequencies obtained using LOBPCG for the test problem Tab. 2.

Phase / Acceleration	$\frac{T_3}{T_2}$	$\frac{T_5}{T_4}$	$\frac{T_3}{T_1}$	$\frac{T_5}{T_1}$
$\mathscr{P}$ = PCG-V	1.3	1.1	2.9	4.7
LOBPCG	1.1	1.1	1.9	2.6

Tab. 4. Acceleration of the LOBPCG implementations in Tab. 1 for various computational scenarios.

this time is reduced even further, resulting in about 1.9 (K40) and 2.6 times (P100) better performance than the reference CPU-only implementation.

The results for all GPU implementations could be further improved by introducing blocking. In fact, a sparse matrix needs to be multiplied by several vectors in various steps of the LOBPCG algorithm. This multiplication is executed on a GPU faster when the vectors are blocked and collected in a single tall and skinny matrix. To solve sparse systems with multiple right-hand sides, we developed our own computational kernels for the sparse matrix–dense matrix product (SpMM). Using the Sliced-ELLR-T format, the Pascal P100 accelerator obtained 112 GFlop/s on a sparse real-valued matrix with sixteen vectors in a block, while no more than 23 GFlop/s was achievable using cuSPARSE [30]. It should however be noted that there is an upper limit on the acceleration of the PCG-V preconditioner. The performance of this solver depends on two factors: (1) the speed of the BLAS1 and sparse matrix multiplications and (2) the direct solution performance on the lowest level of the preconditioner. We used a GPU to accelerate (1), but in all implementations the direct solver is performed using Intel MKL and LU factorization. According to Amdahl's law [31], the maximum acceleration that can be achieved for PCG-V is around 11. The speedups for various scenarios are given in Tab. 4. It can be seen that, with two P100s, we can achieve 4.7, which is a decent result.

The Amdahl's law analysis can also be applied to compute the maximal speedup that can be obtained for PCG-V using two GPUs. With two K40s and P100s performing the computations, the upper bounds are 1.66 and 1.46, respectively. Thus, the speedups of 1.3 and 1.1 obtained in our PCG-V implementation are about 76% and 79% of the theoretical maximal dual-GPU performance. Adding a second GPU gives a slight speed improvement importantly also provides extra GPU memory, allowing larger problems to be solved.

Also, even if we somehow managed to reduce the time taken by the most time-consuming steps of the LOBPCG algorithm (the preconditioning in Step 12 and the null-space filtering in Step 13) in GPU setups to zero, this would subtract 43.1 seconds from the total LOBPCG time for the dual P100 scenario, yielding a maximum speedup of $4.79 = \frac{253.9}{(96.1-43.1)}$. Our current result is 2.6, or 54% of the upper bound.

6. Conclusion

GPU-accelerated solvers for the sparse symmetric generalized eigenvalue problems arising from the FEM simulation of microwave cavities have been described and their performance has been investigated for single and dual GPU hardware setups using two types of GPUs, K40s (NVIDIA Kepler) and P100s (NVIDIA Pascal). The solvers are based on the locally optimal block preconditioned conjugate gra-

dient method (LOBPCG) with null-space filtering. We used the conjugate gradient method with a hierarchical multilevel preconditioner operating on a single V-cycle for the preconditioning of the LOBPCG iterations and for filtration. The speeds of the single and dual-GPU solvers for both types of accelerators were compared to a multithreaded implementation of LOBPCG optimized for a multicore central processing unit (CPU, Intel Xeon E5-2680 v3, twelve cores). We found that for all the tested computational scenarios, the use of the graphics accelerators reduces the time to solution by a factor ranging from 1.8 to 2.6. Further speed gains are expected by using blocking in the sparse matrix-vector multiplication and LDL^{T} factorization of matrices on the lowest level of the PCG-V solver.

Acknowledgments

This work was supported by the Polish National Science Centre under Contract DEC-2014/13/B/ST7/01173 and also, from December 2017 to August 2018, by the Foundation for Polish Science within the TEAM-TECH Programme, cofinanced by the European Regional Development Fund, Smart Growth Operational Programme 2014–2020 (Project EDISOn: Electromagnetic Design of flexIble SensOrs). The Pascal P100 GPU was purchased using statutory funds provided by the Faculty of Electronics, Telecommunications, and Informatics at Gdańsk University of Technology.

References

[1] KRAKIWSKY, S. E., TURNER, L. E., OKONIEWSKI, M. M. Acceleration of finite-difference time-domain (FDTD) using graphics processor units (GPU). In *IEEE MTT-S International Microwave Symposium Digest*. Fort Worth (USA), 2004, p. 1033–1036. DOI: 10.1109/MWSYM.2004.1339160

[2] INMAN, M. J., ELSHERBENI, A. Z. Programming video cards for computational electromagnetics applications. *IEEE Antennas and Propagation Magazine*, 2005, vol. 47, no. 6, p. 71–78. DOI: 10.1109/MAP.2005.1608730

[3] SYPEK, P., DZIEKONSKI, A., MROZOWSKI, M. How to render FDTD computations more effective using a graphics accelerator. *IEEE Transactions on Magnetics*, 2009, vol. 45, no. 3, p. 1324–1327. DOI: 10.1109/TMAG.2009.2012614

[4] DE DONNO, D., ESPOSITO, A., TARRICONE, L., et al. Introduction to GPU computing and CUDA programming: A case study on FDTD [EM programmer's notebook]. *IEEE Antennas and Propagation Magazine*, 2010, vol. 52, no. 3, p. 116–122. DOI: 10.1109/MAP.2010.5586593

[5] DE DONNO, D., ESPOSITO, A., MONTI, G., et al. Parallel efficient method of moments exploiting graphics processing units. *Microwave and Optical Technology Letters*, 2010, vol. 52, no. 11, p. 2568–2572. DOI: 10.1002/mop.25534

[6] DE DONNO, D., ESPOSITO, A., MONTI, G., et al. MPIE/MoM acceleration with a general-purpose graphics processing unit. *IEEE Transactions on Microwave Theory and Techniques*, 2012, vol. 60, no. 9, p. 2693–2701. DOI: 10.1109/TMTT.2012.2203924

[7] 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 Transactions on Antennas and Propagation*, 2014, vol. 62, no. 11, p. 5634–5646. DOI: 10.1109/TAP.2014.2350536

[8] GUAN, J., YAN, S., JIN, J.-M. An OpenMP-CUDA implementation of multilevel fast multipole algorithm for electromagnetic simulation on multi-GPU computing systems. *IEEE Transactions on Antennas and Propagation*, 2013, vol. 61, no. 7, p. 3607–3616. DOI: 10.1109/TAP.2013.2258882

[9] KLÖCKNER, A., WARBURTON, T., BRIDGE, J., et al. Nodal discontinuous galerkin methods on graphics processors. *Journal of Computational Physics*, 2009, vol. 228, no. 21, p. 7863–7882. DOI: 10.1016/j.jcp.2009.06.041

[10] CAPOZZOLI, A., KILIC, O., CURCIO, C., et al. The success of GPU computing in applied electromagnetics. *Applied Computational Electromagnetics Society Journal*, 2018, vol. 33, no. 2. ISSN: 1054-4887

[11] DZIEKONSKI, A., LAMECKI, A., MROZOWSKI, M. GPU acceleration of multilevel solvers for analysis of microwave components with finite element method. *IEEE Microwave and Wireless Components Letters*, 2011, vol. 21, no. 1, p. 1–3. DOI: 10.1109/LMWC.2010.2089974

[12] 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

[13] 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

[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] 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

[16] AURENTZ, J. L., KALANTZIS, V., SAAD, Y. Cucheb: a GPU implementation of the filtered lanczos procedure. *Computer Physics Communications*, 2017, vol. 220, p. 332–340. DOI: 10.1016/j.cpc.2017.06.016

[17] ANZT, H., TOMOV, S., DONGARRA, J. Accelerating the LOBPCG method on GPUs using a blocked sparse matrix vector product. In *Proceedings of the Symposium on High Performance Computing (HPC)*. Alexandria (USA), 2015, p. 75–82.

[18] KREUTZER, M., ERNST, D., BISHOP, A. R., et al. Chebyshev filter diagonalization on modern manycore processors and GPGPUs. In *Proceedings of the International Conference on High Performance Computing*. Frankfurt (Germany), 2018, p. 329–349. DOI: 10.1007/978-3-319-92040-5_17

[19] RODRIGUES, W., PECCHIA, A., DER MAUR, M. A., et al. A comprehensive study of popular eigenvalue methods employed for quantum calculation of energy eigenstates in nanostructures using GPUs. *Journal of Computational Electronics*, 2015, vol. 14, no. 2, p. 593–603. DOI: 10.1007/s10825-015-0695-z

[20] DZIEKONSKI, A., REWIENSKI, M., SYPEK, P., et al. GPU-accelerated LOBPCG method with inexact null-space filtering for solving generalized eigenvalue problems in computational electromagnetics analysis with higher-order FEM. *Communications in Computational Physics*, 2017, vol. 22, no. 4, p. 997–1014. DOI: 10.4208/cicp.OA-2016-0168

[21] RUBIO, J., ARROYO, J., ZAPATA, J. Analysis of passive microwave circuits by using a hybrid 2-D and 3-D finite-element mode-matching method. *IEEE Transactions on Microwave Theory and Techniques*, 1999, vol. 47, no. 9, p. 1746–1749. DOI: 10.1109/22.788618

[22] ZHU, Y., CANGELLARIS, A. C. *Multigrid Finite Element Methods for Electromagnetic Field Modeling*. John Wiley & Sons, 2006. ISBN: 9780471741107

[23] INGELSTROM, P. A new set of H (curl)-conforming hierarchical basis functions for tetrahedral meshes. *IEEE Transactions on Microwave Theory and Techniques*, 2006, vol. 54, no. 1, p. 106–114. DOI: 10.1109/TMTT.2005.860295

[24] KNYAZEV, A. V. Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method. *SIAM Journal on Scientific Computing*, 2001, vol. 23, no. 2, p. 517–541. DOI: 10.1137/S1064827500366124

[25] ARBENZ, P., GEUS, R. Multilevel preconditioned iterative eigensolvers for Maxwell eigenvalue problems. *Applied Numerical Mathematics*, 2005, vol. 54, no. 2, p. 107–121. DOI: 10.1016/j.apnum.2004.09.026

[26] DZIEKONSKI, A., LAMECKI, A., MROZOWSKI, M. A memory efficient and fast sparse matrix vector product on a GPU. *Progress in Electromagnetics Research*, 2011, vol. 16, p. 49–63. DOI:10.2528/PIER11031607

[27] DZIEKONSKI, A., SYPEK, P., LAMECKI, A., et al. Communication and load balancing optimization for finite element electromagnetic simulations using multi-GPU workstation. *IEEE Transactions on Microwave Theory and Techniques*, 2017, vol. 65, no. 8, p. 2661–2671. DOI: 10.1109/TMTT.2017.2714670

[28] REWIENSKI, M., DZIEKONSKI, A., LAMECKI, A., et al. A stabilized complex LOBPCG eigensolver for the analysis of moderately lossy EM structures. *IEEE Microwave and Wireless Components Letters*, 2018, vol. 28, no. 1, p. 7–9. DOI: 10.1109/LMWC.2017.2771289

[29] 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, no. 4, p. 970–978.

[30] DZIEKONSKI, A., MROZOWSKI, M. Block conjugate-gradient method with multilevel preconditioning and GPU acceleration for fem problems in electromagnetics. *IEEE Antennas and Wireless Propagation Letters*, 2018, vol. 17, no. 6, p. 1039–1042. DOI: 10.1109/LAWP.2018.2830124

[31] AMDAHL, G. M. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the AFIPS Spring Joint Computer Conference*. Atlantic City (USA), 1967, p. 483–485.

About the Authors ...

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 parallelizing computations for 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.

Michal MROZOWSKI received M.Sc. and Ph.D. degrees, both with honors, from Gdansk 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, and Director of the Center of Excellence for Wireless Communication Engineering (WiComm). He is a Fellow of IEEE.