DOI 10.1007/s10598-022-09545-2
Computational Mathematics and Modeling, Vol. 32, No. 4, October, 2021

BENCHMARKS OF CUDA-BASED GMRES SOLVER FOR TOEPLITZ AND HANKEL MATRICES AND APPLICATIONS TO TOPOLOGY OPTIMIZATION OF PHOTONIC COMPONENTS

Iu. B. Minin,[1,2] **S. A. Matveev,**[1,3,4] **M. V. Fedorov,**[1,5]
I. E. Zacharov,[1] **and S. G. Rykovanov**[1]

Generalized Minimal Residual Method (GMRES) was benchmarked on many types of GPUs for solving linear systems based on dense and sparse matrices. However, there are still no GMRES implementation benchmarks on Tesla V100 compared to GTX 1080 Ti ones or even for Toeplitz-like matrices. The introduced software consists of a *Python* module and a C++ library which enable to manage streams for concurrent computations of separated linear systems on a GPU (and GPUs). The GMRES solver is parallelized for running on a NVIDIA GPGPU accelerator. The parallelization efficiency is explored when GMRES is applied to solve (Helmholtz equation) linear systems based on the use of Green's Function Integral Equation Method (GFIEM) for computing electric field distribution in the design domain. The proposed implementation shew the maximal speedup of 55 ($\bar{t} = 0.017$ s) and of 125 ($\bar{t} = 0.77$ s) for 1024×1024 (on GTX 1080 Ti) and 8192×8192 (on Tesla V100) dense Toeplitz matrices generated from GFIEM. 1024×1024 resolution provides accuracy 6.1% that can be acceptable according to testing and demonstrating on gradient computations and topology optimization. We open up possibilities for robust topology optimization of passive photonic integrated components. That has the advantage, e. g., of faster and more accurate designing photonic components on a PC without a supercomputer.

Keywords: GPU, performance, *CUDA* C++, parallelization, GMRES, Toeplitz-like matrix, Python module, Helmholtz equation, topology optimization.

1. Introduction

The paper presents an implementation of novel GMRES benchmarks for Toeplitz-like matrices generated when resolving the discretized Helmholtz equation running on the Tesla V100 Graphics Processing Unit (GPU). The well-known discretization techniques of the Helmholtz equation [1] usually generate linear system based on Toeplitz matrix. E.g., that can be in utilizing uniform grid for electric field distribution. If one exploits FFT convolution for fast multiplications of Toeplitz-like matrix by vector (matvecs) [2, 3] a solution to linear system can have computational cost order $\mathcal{O}(nm \log n)$, where n is a matrix dimension size and n_G is the number of GMRES iterations.[6]

Popular iterative method for solving linear systems GMRES was introduced by Marchuk and Kuznetsov in 1968 in a geometric version [10]. The algebraic version of this method was proposed by Saad and Schultz

[1] Skoltech Center for Computational and Data-Intensive Science and Engineering, Skolkovo Institute of Science and Technology, Moscow, Russia.
[2] Fryazino Branch of Kotel'nikov Institute of Radio-Engineering and Electronics of Russian Academy of Sciences, Fryazino, Moscow, Russia.
[3] Faculty of Computational Mathematics and Cybernetics, Lomonosov Moscow State University, Moscow, Russia.
[4] Marchuk Institute of Numerical Mathematics, Russian Academy of Sciences, Moscow, Russia.
[5] Sirius University of Science and Technology, Sochi, Krasnodar Krai, Russia.

[6] In general, we assume $n \gg n_G$ (large matrix), otherwise the computational cost of the GMRES orthogonalization has to be included into the complexity order estimation.

Table 1
Parallel GMRES on Different GPUs: Number of Non-Zero Elements $n_{\bar{z}}$, Computational Time t, Maximal Speedup s over Sequential Run on the CPU

GMRES performance on different GPU systems

GPU	Data type	Data size	t [seconds]	s	CPU	
1×GTX 1080	BEM elements	7568 elements	50	90	Core i7-6800K	[3]
1×Tesla C2070	sparse matrix	$n \approx 2 \cdot 10^5$; $n_{\bar{z}} \approx 3 \cdot 10^6$	0.14	21	Xeon E5620	[5]
1×Tesla C2070	sparse matrix	$n \approx 2 \cdot 10^6$; $n_{\bar{z}} \approx 3 \cdot 10^7$	6	11	Xeon X5570	[6]
1×Tesla M2070	sparse matrix	$n \approx 9 \cdot 10^5$; $n_{\bar{z}} \approx 2 \cdot 10^6$	0.98	20	Xeon E5	[7]
3×Tesla M2090	sparse matrix	$n \approx 6 \cdot 10^5$; $n_{\bar{z}} \approx 4 \cdot 10^6$	0.5	31	Xeon E5	[8]
12×Tesla C1060	sparse matrix	$n \approx 3.4 \cdot 10^6$; $n_{\bar{z}} \approx 4.4 \cdot 10^8$	0.7	201	Xeon E5530	[9]

in 1986. The method is based on solving the least squares problem for Krylov subspace at each step of the iteration. The method is usually provided with the Arnoldi iteration [11, 12].

Execution of GMRES can be accelerated with the use of restarts. That reduces computer memory consumption and enhances the convergence properties, which in practice often benefits in terms of computational time. GMRES with restarts is similar to the well-known Anderson acceleration [11, 13].

The main computational cost of GMRES is contained in the matvec problem when linear system is solved for large matrices because all (small) subsidiary (for GMRES) matrices are of sizes within $(m+1) \times (m+1)$ due to much less computational times for small matrix arithmetic. Computational cost for dense matrix matvec is $\mathcal{O}(n^2)$. However, matvec problem for Toeplitz-like matrices and mixed matrices can be solved by using the convolution based on the Fast Fourier Transforms (FFTs). Thus, computational cost of an FFT-based matvec can be $\mathcal{O}(n \log n)$ instead of $\mathcal{O}(n^2)$ [2, 3].

Computational stability of GMRES depends on its implementation taking into account the round-off errors. The round-off errors arise in GMRES when creating the orthogonal basis and, in particular, computing the residual vector (the operation to subtract) or normalizing the orthogonal basis vectors (the operation to divide) [14–16].

GMRES was parallelized on GPUs [3, 5, 6] and CPUs [17–19]. GMRES parallelization studies show implementation efficiency increase with the characteristic size of matrix (and sometimes with the computational time). The computational time is determined not only by the computational cost of the method, but by the parallelization opportunities as well.

Single-GPU and **multi-GPU** parallel implementations of GMRES are compared in Table 1. While in general there is a substantial increase in performance for GPU-based tests we propose the implementation can be improved. In addition, we propose to demonstrate GMRES performance quantities for Tesla V100.

Main problem in these GMRES implementations is time consumption spent on data transfers between RAM and GPUs. E.g., the data transfer bandwidth is limited on the PCIe 3.0 x16 bus to 12.5 GB/s (as could be measured by the NVIDIA "bandwidthTest" tool), which is usually significantly lower than computational processing of data. E.g., computation bandwidth of a (single) Tesla M2050 GPU is $\approx 18.8 \div 105.2$ GB/s when computing a matrix transpose by using its different implementations [20]. Therefore, the time expended on the GPU data transfers can be higher than the computational time on the GPU. That means total time spent on data transfer from RAM to GPU and back is $1.5 \div 8.6$ times bigger than time spent on computing on the GPU. With number of processes utilizing PCIe bus, partial bandwidth decreases that usually leads to decrease in total performance [21, 22].

Data transfer in GMRES is implemented with the memory copy operations [3, 5, 8, 9]. They constitute the communication latency limiting the speedup. That fact was noticed for the Krylov methods (such as GMRES) [23]. Thus, performance can be improved by $\approx 50\%$ [24, 25]. Our work proposes copy-free GMRES implementation.

To speed up GMRES, the algebraic operations of the algorithm were studied to evaluate and optimize the most time-consuming parts of GMRES implementation [6, 8, 26].

GMRES is often used to solve the Helmholtz equation [3, 27]. These solutions can be used, e.g., for the topology optimization of wave devices [28] and can therefore benefit greatly from parallelization of GMRES (see *CUDA* C++ implementations [29, 30]). Thus, we exploited this approach for benchmarks in this paper.

In the previous work, we demonstrated results on topology optimization based on *Python* GFIEM implementation [28]. This work is improvement of the previous one via *CUDA* C++ parallelization of GMRES solver for GFIEM. The presented examples can be considered as a good benchmark of the implementation of GMRES. The speedup, run-time and convergence of GMRES are the main metrics to judge the GMRES implementation.

This paper possesses four purposes:

(i) to first represent benchmarks (computational times, speedups) of Tesla V100 GPU (compared to ones of GTX 1080 Ti ones and of Xeon Gold 6140 CPU) on a GMRES implementation (Section 5);

(ii) to describe distinctive features of first GMRES implementation (Section 3) for solving linear systems based on Toeplitz-like matrices (Toeplitz, Hankel, circulant or mixed ones) (Section 2);

(iii) to investigate the graph of accuracy of GFIEM (w. r. t. analytical Mie solution to scattering of electromagnetic waves by a homogeneous cylinder) versus number of nodes (resolution) when solving by means of proposed GMRES implementation (Sections 4, 5);

and

(iv) to test topology optimization (TO) based on the proposed GFIEM implementation and to find out acceptable values of accuracy, of number of nodes (resolution) and of wavelength per domain for TO feasibility (Sections 4, 5).

This research reports the efficiency of the parallelization method applying GMRES to the Helmholtz linear system based on the use of GFIEM [1] (as a test of the proposed GMRES implementation work) for computing electric field distribution and for designing new photonic components (Section 6).

Program packages such as Trilinos, PETSc, Ginkgo have GMRES implementations. However, they do not take into account Toeplitz-like matvec FFT-acceleration opportunities [31–33]. This research does not investigate any new numerical aspects for GMRES, The research explores GMRES implementation issues. The novel GMRES implementation is based on known numerical techniques. However, there is still no paper proposed FFT-based approach to accelerate GMRES computations and tested real program package for solving linear systems based on Toeplitz-like matrices.

2. Numerical Formulation

A linear system is given by:

$$Ax = b, \tag{1}$$

where we assume the square linear system matrix A is nonsingular, otherwise GS-GMRES may break down before the method converges [11, 34]. A solution to singular linear system can be obtained by using the breakdown-free GMRES (BF-GMRES) [34] and range restricted GMRES (RR-GMRES) [35]. Therefore, in the next sections we have included the information on the proposed Toeplitz-like matvecs.

Toeplitz-Like Matrix. In this paper, A is Toeplitz-like matrix if that can be obtained from square finite Toeplitz matrix by changing order of rows [36, 37]. Thus, computational complexity of this matvec coincides with Toeplitz matvec complexity $\mathcal{O}(n \log n)$, in contrast to computational complexity of dense matrix matvec equal to $\mathcal{O}(n^2)$, because the most efficient way to multiply Toeplitz matrix by a vector is to use FFTs to matvec [2, 3]. In addition, computational complexity of operation to reorder matrix can be equal to $\mathcal{O}(\sqrt{n} \log n)$ [38] that is negligible compared to $\mathcal{O}(n \log n)$. Examples of Toeplitz-like matrices are Hankel, Circulant [39].

Mixed Matrix. In this paper, A is mixed matrix if

$$A = \alpha D + \beta R, \tag{2}$$

where

D — diagonal matrix,

R — Toeplitz-like matrix,

α and β — real nonzero numerals.

3. Program Descriptions

The implementation codes are available on github [40, 41]. They contain sequential and parallel versions of GMRES implemented in *CUDA* C++ and *Python*.

Efficiency Implementation Improvement (Miscellaneous Distinctive Features). Consider two consecutive iterations of GMRES (listing 1). The subsidiary matrices of the next iteration have one more columns and one more rows. The previous subsidiary matrices are similar in structure. Furthermore, first Jtotal matrix 3 and Givens rotation matrix 4 are almost identity matrices (four elements of each one are set with two values). Thus, the subsidiary matrices are filled recursively in structure with GMRES iteration. This is the basis for optimizing the storage and use of subsidiary matrices. That provides, e.g., less GPU memory utilization. Thus, we take this fact into account. These opportunities were covered for the first time ever. That is one of the distinctive features of GMRES implementation in this paper compared to others.

Proposed GMRES Implementation Description. The GMRES implementation (listing 1) starts with allocating memory (the malloc $(V,\ I,\ r)$ function) for all support matrices $V,\ I,\ H,\ \tilde{H},\ J,\ \tilde{J},\ \alpha,\ \beta,\ P$ and residuals of the algorithm r in a single function call. The function pointer_shift $(V,\ I,\ r)$ sets pointers of the support matrices by cutting the allocated memory into support matrices. Memory allocation is conducted by three variables: V (*complex* array), I (*integer* array), and r (*float* array). In this way the number of memory allocation procedures was minimized. Same pointer_shift $(V,\ I,\ r)$ sets the pointers of the elements $\tilde{J}_0,\ \tilde{J}_1,\ \tilde{J}_2,\ \tilde{J}_3$ of the Givens rotation matrix $\tilde{J}$. Complex coefficients $\alpha \equiv 1$ and $\beta \equiv 0$ are used in *cuBLAS* matrix multiplications, i.e.,

$$C = \alpha AB + \beta C \equiv AB.$$

Some supporting functions of implementation (listing 1) are listed (listing 2). The function set_first_Jtotal (H)

Algorithm 1. Gram–Schmidt GMRES (GS-GMRES).

```
1: procedure GMRES($A$, $x_0$, $output$)
2:     malloc($V$, $I$, $r$)
3:     $H$, $\tilde{H}$, $J$, $\tilde{J}$, $\alpha$, $\beta$, $\tilde{J}_0$, $\tilde{J}_1$, $\tilde{J}_2$, $\tilde{J}_3$, $P \leftarrow$
4: $\leftarrow$ pointer_shift($V$, $I$)
5:     $\Delta b \leftarrow$ diff_matvec ($A$, $x_0$, $b$)
6:     $r_0 \leftarrow \|\Delta b\|$
7:     $condition \leftarrow (tol < r_0)$
8:     $V_0 \leftarrow \Delta b / r_0$
9:     $\alpha \leftarrow 1 + 0\,\mathbf{i}$
10:    $\beta \leftarrow 0 + 0\,\mathbf{i}$
11:    if $condition = \mathbf{true}$ then
12:        memset($H$, 0)
13:        $w \leftarrow$ matvec ($A$, $V_0$)
14:        $H_{00} \leftarrow V_0\, w^{\mathbf{H}}$: matrix operator $\mathbf{H}$ means to conjugate and transpose matrix
15:        $w \leftarrow w - H_{00}\, V_0$
16:        $H_{10} \leftarrow \mathrm{norm}(w)$
17:        $\gamma \leftarrow 1/H_{10}$
18:        $w \leftarrow \gamma\, w$
19:        $J \leftarrow I$: Initialize Given's rotation matrix $J$ as identity matrix $I$
20:        $J \leftarrow$ set_first_Jtotal($H$)
21:        $condition \leftarrow$ check($J$, $\jmath$, $r$, $tol$)
22:        $\jmath \leftarrow 1$
23:        while $\jmath < m$ & $condition = \mathbf{true}$ do
24:            $w \leftarrow$ matvec ($A$, $V_\jmath$)
25:            for $\imath = \overline{0,\, \jmath}$ do
26:                $H_{\imath\jmath} = V_\imath\, w^{\mathbf{H}}$
27:                $w \leftarrow w - H_{\imath\jmath}\, V_\imath$
28:            $H_{\jmath+1,\jmath} \leftarrow \mathrm{norm}(w)$
29:            $\gamma \leftarrow 1/H_{\jmath+1,\,\jmath}$
30:            $w \leftarrow \gamma\, w$
31:            $\tilde{H} = \alpha J\, H + \beta \tilde{H}$
32:            $\tilde{J} \leftarrow$ set_4_GRM_elements($\tilde{H}$)
33:            $J = \alpha \tilde{J}\, J + \beta J$
34:            $condition \leftarrow$ check($J$, $\jmath$, $r$, $tol$)
35:            $\jmath \leftarrow \jmath + 1$
36:        $H \leftarrow \alpha J\, H + \beta H$
37:        $c_\ell \leftarrow J_{\ell 0}\ \ \forall \ell = \overline{0,\, \imath - 1}$
38:        if $\jmath = 1$ then
39:            $c \leftarrow c/H$
40:            $x_0 \leftarrow x_0 + V_0\, c$
41:        else
42:            $c \leftarrow$ solve_small_LS($H$, $c$)
43:            for $\imath = \overline{0, \jmath - 1}$ do
44:                $x_0 \leftarrow x_0 + V_\imath\, c_\imath$
45:    free($V$, $I$)
```

Algorithm 1 (continued).

```
46: function check($J$, $\jmath$, $r$, $tol$)
47:     $r_{\jmath+1} \leftarrow r_0 \, |J_{\jmath+1,\,0}|$
48:     $condition \leftarrow (tol < r_{\jmath+1})$
49:     return $condition$
50: procedure solve_small_LS($H, c$): solving small linear system
51:     cgetrf $(H,\, P)$: LU factorization
52:     cgetrs $(H,\, P,\, c)$: computing the solution
53: function diag($D$)
54:     $d_\imath \leftarrow D_{\imath\imath} \quad \forall \imath = \overline{0,\, N-1}$
55:     return $d$
```

Algorithm 2. Matrix by vector multiplications (matvecs).

```
1: function matvec_A($A$, $x$)
2:     $y \leftarrow Ax$
3:     return $y$
4: function diff_matvec_A($A$, $x$, $b$)
5:     $y \leftarrow$ matvec_A$(A,\, x)$
6:     $\Delta b \leftarrow y - b$: for all elements of $y$
7:     return $\Delta b$
8: function matvec_T($T$, $x$)
9:     $T^e \leftarrow$ extend_T$(T)$
10:    $x^e \leftarrow$ extend_vec$(x)$
11:    $y^e \leftarrow$ IFFT (FFT $(T^e) *$ FFT $(x^e)$): operator "$*$" means Kronecker product.
12:    return $y^e$
13: function diff_matvec_T($T$, $x$)
14:    $y^e \leftarrow$ matvec_T$(T,\, x)$
15:    $\Delta b \leftarrow y^e - b$: for not all elements of $y^e$, i.e., for $\imath = \overline{0,\, N-1}$, $\jmath = \overline{0,\, N-1}$
16:    return $\Delta b$
17: function matvec_DpT($D$, $T$, $x$)
18:    $y_1^e \leftarrow$ matvec_T$(T,\, x)$
19:    $y_2 \leftarrow$ diag$(D) * x$
20:    $y \leftarrow y_1^e + y_2$: for $\imath = \overline{0,\, N-1}$, $\jmath = \overline{0,\, N-1}$
21:    return $y$
22: function diff_matvec_DpT($D$, $T$, $x$, $b$)
23:    $y_1^e \leftarrow$ matvec_T$(T, x)$
24:    $y_2 \leftarrow$ diag$(D) * x$
25:    $\Delta b \leftarrow y_1^e + y_2 - b$: for $\imath = \overline{0,\, N-1}$, $\jmath = \overline{0,\, N-1}$
26:    return $\Delta b$
```

gives $(m+1) \times (m+1)$ sparse matrix (3), where $\zeta = H_{00}/\theta,\ \vartheta = H_{10}/\theta,\ \theta \equiv \sqrt{H_{00}^2 + H_{10}^2}$:

$$\begin{bmatrix} \zeta & \vartheta & & & & & \\ -\overline{\vartheta} & \zeta & & & \text{0} & & \\ & & 1 & & & & \\ & & & 1 & & & \\ & & & & 1 & & \\ & \text{0} & & & & \ddots & \\ & & & & & & 1 \end{bmatrix}. \tag{3}$$

Note that the function set_4_GRM_elements $(\tilde{H})$ sets only 4 elements of $m \times m$ Givens rotation matrix $\tilde{J}$ (4): $\tilde{\zeta} = \tilde{H}_{\jmath\jmath}/\tilde{\theta},\ \tilde{\vartheta} = \tilde{H}_{\jmath+1,\jmath}/\tilde{\theta},\ \tilde{\theta} \equiv \sqrt{\tilde{H}_{\jmath\jmath} + \tilde{H}_{\jmath+1,\jmath}}$ [39], where $\tilde{J}_0 = \tilde{J}_3 = \tilde{\zeta},\ \tilde{J}_2 = -\overline{\tilde{J}_1} = \overline{\tilde{\vartheta}}$ in

$$\begin{bmatrix} 1 & & & & & & \\ & \ddots & & & \text{0} & & \\ & & 1 & & & & \\ & & & 1 & & & \\ & & & & 1 & & \\ & \text{0} & & & & \tilde{\zeta} & \tilde{\vartheta} \\ & & & & & -\overline{\tilde{\vartheta}} & \tilde{\zeta} \end{bmatrix}. \tag{4}$$

The function free $(V,\ I)$ releases all memory allocated for the support matrices. Therefore, the number (computational time) of memory operations was minimized.

Subresults. We measured computational time efficiency for subsidiary matrix computations: matrix-by-matrix multiplications, AxPY operations, allocating and deallocating blocks of memory on different by using implementations. E.g., operation $B^T A^T = C^T$ (computational time can equal $20\mu s$) when applying the transpose flags is (20%) faster than the same implementation by executing $AB = C$ $(13\mu s)$ with the use of C — transposing $(11\mu s)$. Furthermore, residual computation (operation of multiplying residual vector nor by norm of Givens rotation matrix element. The operation is described in **function** check in listing 1) by using GPU $(25\mu s)$ is (150%) more time-consuming than by using CPU with the migration of data from GPU to CPU and back $(10\mu s)$.

Subconclusion. We reused the memory by using new implementation by avoiding copying GPU data as well as matrix transfers between GPU and CPU. In addition, we researched new opportunities to increase total speedup of the program by (minimizing the computational time of each operation) equating parallel operations of speedup less than one with sequential operations. We investigated the computational times and speedups of each operation. Thus, the proposed implementation has additional distinct feature in less time and memory consumptions. The total efficiency is proposed to be improved by 15%.

Toeplitz-like Matvec Compared to Dense Matrix Matvec. If Toeplitz-like matrix A is, e.g., of sizes $n^2 \times n^2$ (where n may be equal to $2^\nu,\ \nu \in \mathbb{N}$). Number of elements of such a matrix equals $N = n^2 = 2^{4\nu}$. Thus, its surrogate $G \in \mathbb{C}^{(2n-1)\times(2n-1)}$ $(\mathbb{C}^{(2^{\nu+1}-1)\times(2^{\nu+1}-1)})$ has $\tilde{N} = (2n-1)^2$ elements that is $\approx n^2$ $(\approx 2^{2\nu})$ (for $\log_2(n) \gg 1$) less than for matrix A. That provides advantage of $\approx n^2$ $(\approx 2^{2\nu})$ less memory utilization and computational time. Furthermore, the Toeplitz-like matrix surrogate allows to accelerate computations. Moreover, that can be stored in GPU memory when that is reused many times like in proposed examples 4.

Furthermore, if n can be represented as a^{η}, where a is prime number and $\eta \in \mathbb{N}$, then FFT can be substituted by prime-factor (Good–Thomas) algorithm (PFA).

In addition, we reused variables in order to to achieve GMRES memory utilization reduction of $\approx -\dfrac{100}{n_G + 7}\%$ (if $n \gg n_G$), where number of GMRES iterations n_G that is usually much less than characteristic size n of surrogate matrix. E.g., in this paper, we used $n = 1024$ and $n_G = 30; 50$. Thus, GPU memory was reduced by 2.7% and 1.8% (as examples). That is additional distinctive feature of this GMRES implementation.

Sequential Version. Sequential version of GS-GMRES is implemented in a single-threaded *MKL* C++, i.e., all operations of GMRES implementation are sequential and executed one after another [41].

Parallel Version. Parallel version of GS-GMRES is implemented in *CUDA* C++. This implementation benefits from data parallelism by conducting the matrix operations provided by kernels from *CUDA* C++ libraries: *cuBLAS*, *cuFFT* and *cuSOLVER*. The implementation synchronizes itself, it checks the GPU computations for accuracy after essential operations to provide information about the validity of the results [41].

Python Module. *CUDA* C++ GMRES implementation is wrapped into *Python* module by (automatically) creating *CUDA* C++ *shared object file*. Moreover, the implementation provides capability to manage different streams on a GPU (and GPUs) for parallel computations of separated linear systems based on Toeplitz-like matrices [40].

4. Application Examples

The Helmholtz Equation. We solve the two-dimensional Helmholtz equation for component material distribution (permittivity ε) in the medium $(\tilde{\varepsilon})$ that connects total, incident fields $u(r), w(r) \subset \mathbb{C}$:

$$\begin{cases} \triangle u(r) + k_0^2\, u(r) = 0, & r \in \mathbb{R}^2 \backslash \Omega; \\ u(r) = 0, & r \in \partial\Omega\text{: } \textit{Dirichlet boundary c}; \\ \lim\limits_{|r|\to\infty} (\mathbf{i}k_0(u - w)(r) - \partial_{|r|}(u - w)(r))|r| = 0 : & \textit{Sommerfeld radiation c}; \end{cases} \tag{5}$$

where $\triangle$ is Laplace operator, r is space vector and k_0 is a wave number [1]. The Helmholtz equation solution can be represented for uniform grid of square two-dimensional space Ω through two-level Toeplitz matrix $H^{(2)} \in \mathbb{C}^{n^2 \times n^2}$ [1, 28, 42] in linear system matrix A:

$$Ax = Ix - k_0^2\, (\varepsilon - \tilde{\varepsilon})\, H^{(2)}\, m * x, \tag{6}$$

where $x \in \mathbb{C}^{n^2}$ is electric field distribution for uniform grid $\tilde{r} \in \mathbb{R}^{n \times n}$ of Ω, I is the identity matrix, operator "$*$" is Kronecker product and $m \in \{0; 1\}^{n^2}$ is a binary mask of photonic component material distribution in the design domains. This is detailed in the previous work [28], where we utilized FFT for multiplication of Toeplitz matrix $H^{(2)}$ by vector. In addition, number of computations was reduced by using two-level Toeplitz matrix surrogate $G \in \mathbb{C}^{(2n-1)\times(2n-1)}$ [2, 28]. Number of Ω nodes is equal to n^2.

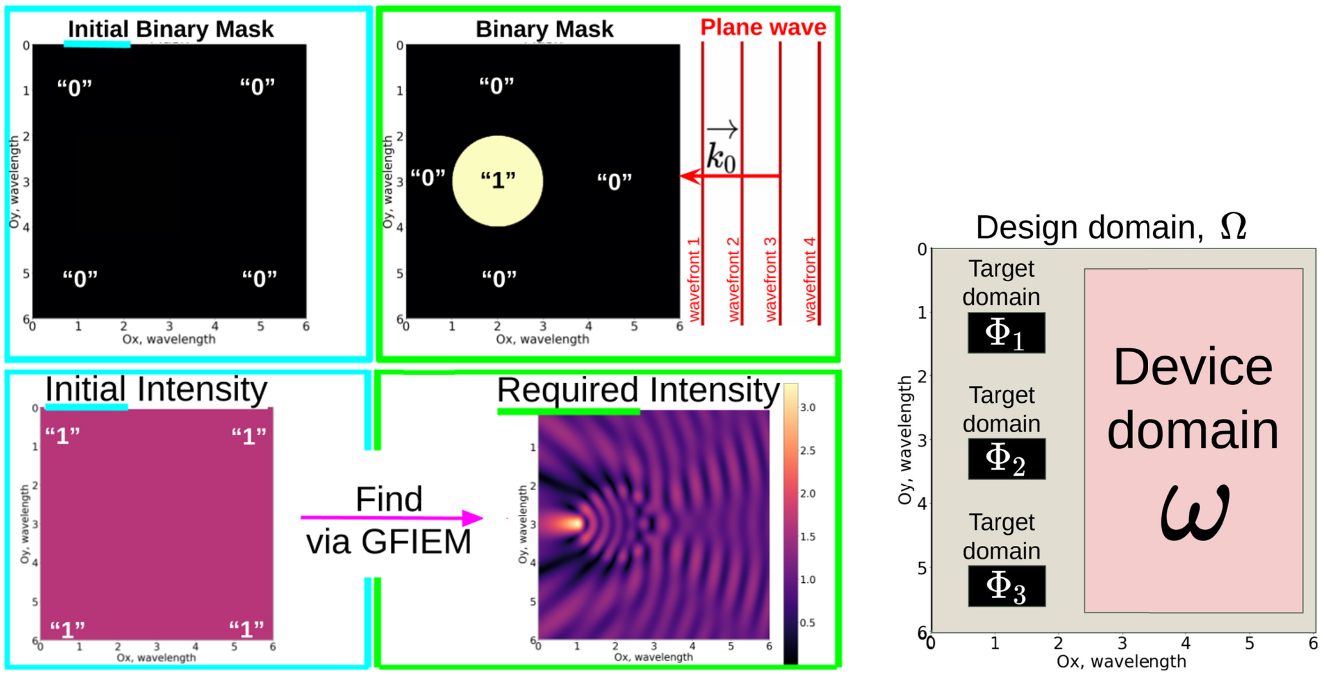


Fig. 1. GFIEM conversions (on the left) and TO domains of three-focal glass superlens (on the right).

E.g., consider a simple problem when glass $(\varepsilon = 2.25)$ nanotube of infinite length is under only one monochromatic polarized plane wave in air $(\tilde{\varepsilon} = 1)$ in domain Ω of sizes $6\lambda \times 6\lambda$ when resolution equals $n/6$ nodes per wavelength λ. The nanotube is specified by circular binary mask m (Fig. 1). The challenge is to find the intensity distribution for this device. We start with using the initial (zero) mask which represents lack of photonic component material substance (glass) in its medium (air). The field distribution is well-known for zero mask. By using this well-known initial field, the field distribution for any binary (non-zero) mask m can be computed using the Green's Function Integral Equation Method (GFIEM) [1, 28]. This is shown in Fig. 1; the initial field can be, for example, sinusoidal [43].

Topology Optimization (TO). Topology optimization (TO) is a mathematical optimization problem that involves designing the shape and changing the structure of the device so that the superposition of the target functions has a downward trend [44]. TO started being used for photonic components in the last century, for example, for wavelength-division multiplexers [45]. In addition, it has been proposed for other devices such as the photonic rings [46]. Attempts were made to accelerate TO for inverse design of photonic crystals (accuracy is high, computational time is 1–10 hours) [47] as well as for photonic structures using machine learning (computational time is milliseconds, but accuracy is sufficiently lower) [48]. However, there is no commercially available or widely used approach that combines the advantages of both methods. We propose a novel approach which has a time of one TO iteration of the order of 1 minute for measurement accuracy of the order of 1%. Moreover, it will be intended for optimizing photonic components.

The novelty of this approach consists in applying Green's Function Integral Equation Method (GFIEM) (Helmholtz equation solver) for electric field distribution computation. GFIEM is represented by linear system with core matrix that is Toeplitz-like [1]. That linear system can be solved numerically by using the implementation of the Generalized Minimal Residuals algorithm (GMRES) [28], which was GPGPU-accelerated in the C programming language [41] and wrapped in the Python software package [40]. Other accelerated versions of GMRES were also introduced earlier [9]. However, they do not take into account the opportunity of Toeplitz-like

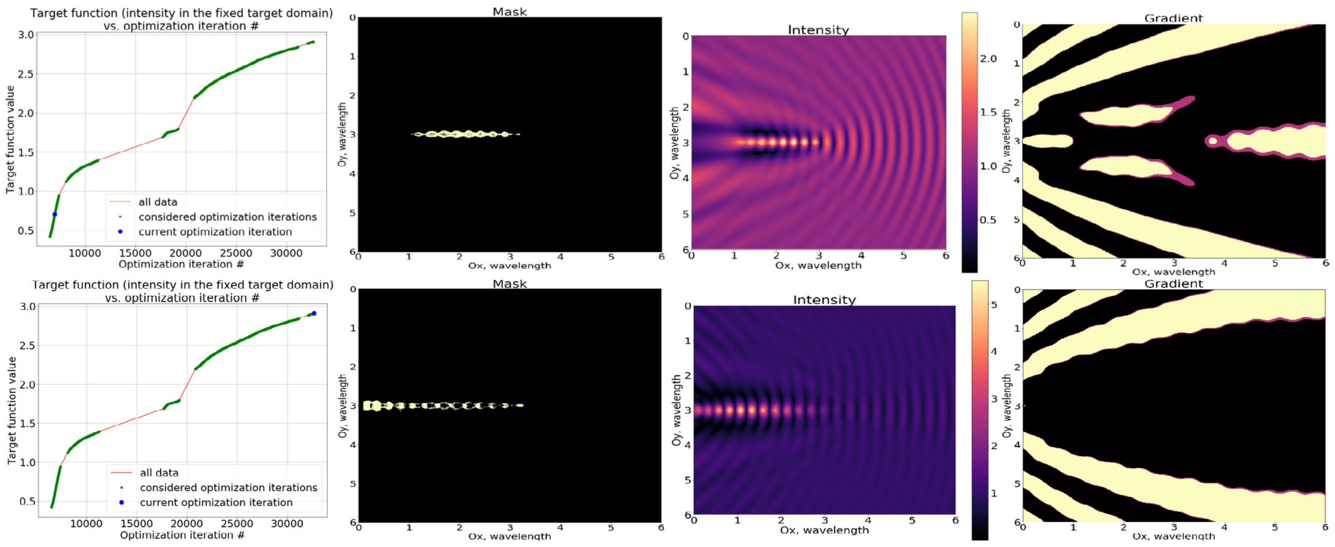


Fig. 2. Topology optimization of one-focal glass lens.

matvec FFT-acceleration [2]. Thus, this additional distinctive feature is implemented in the proposed approach and its software package.

Analysis of world experience shows reducing the cost and increasing the competitiveness of devices and equipment are primarily associated with the transition from electronic integrated circuits (ICs) to photonic circuits (such as PICs) as well as with TO application.

The difficulty is caused by investigating an acceptable ratio of the computational time to the achieved accuracy. Computations of objective functions and predicting their changes are the most time-expensive. Moreover, the proposed implementation is based on solving the Helmholtz equation as a linear system based on a Toeplitz-like matrix. The solution of linear systems can be achieved in a minimal amount of computational time using GMRES [11]. Furthermore, the most computationally expensive part of it is matrix-vector products, which can be accelerated using FFT, because the core matrix of the GFIEM method is Toeplitz-like [2].

There is a problem to optimize structures of, e.g., antennas, splitters, Bragg reflectors, and fiber Bragg gratings under superposition of polarized monochromatic waves in order to increase resolution in subwavelength image formation [28].

There are: homogeneous device domain (represented by black-to-white binary mask), point sources of resultant wave (that are infinitely far from the device), and target image formation (represented by intensity distributions in target domains) (e.g., proposed TO domains for three-focal glass superlens is shown in Fig. 1).

The proposed software introduces groundwork for TO approach. This approach promotes a faster development of photonic component and enables to accelerate a photonic component fabrication, reduce its size as well as the cost. In general, one of main objectives of the photonic component topology optimization is to find such optimal shape for device that would give intensity distributions (to be closest to the required ones) in target domains. For example, target function for superlenses is maximization of intensities in their target domains.

Consider an instrument to compute target functions. Target functions of these nanophotonic systems can be described, e.g., by Helmholtz equation in the matrix form (6). In structural optimization method, it is essential to utilize as fast solver as possible in order to compute the target functions. That is why we use fast converging stable GFIEM to compute intensity distribution. That is the VIE method [1, 28].

Superlens topology optimization results in the considered design domain Ω are represented in two quadro-figures (after 7000 and 32653 topology optimization iterations) (Fig. 2). Each quadro-figure shows target function

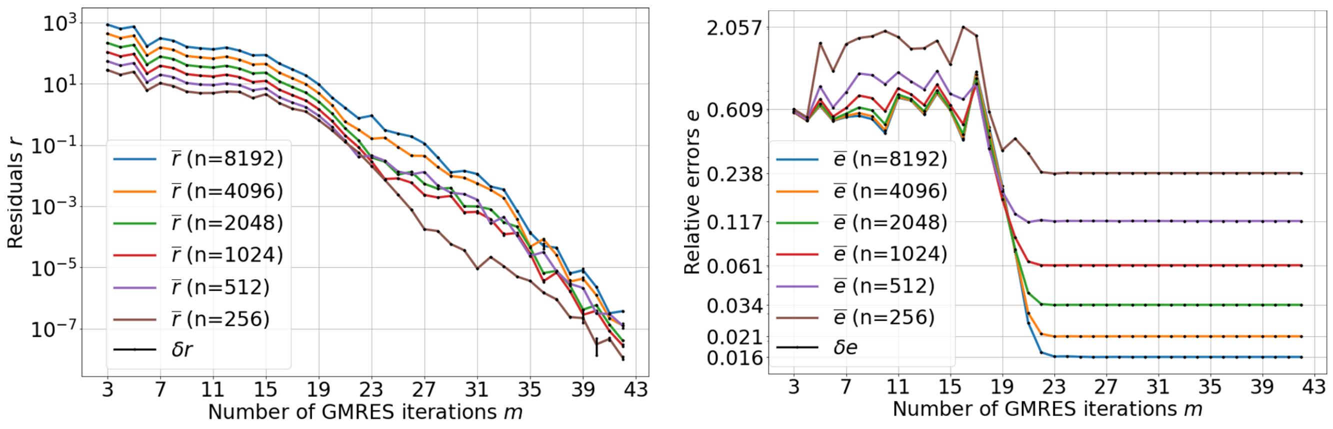


Fig. 3. *CUDA* GMRES residuals (left) and relative errors (right).

versus optimization iteration number graph, binary mask distribution, intensity distribution and intensity gradient (w.r.t. mask m) distribution. The current optimization iteration on target function curve is marked in blue. The white and black pixels of the gradient distribution represent positive and negative mask gradients of intensity distribution while red pixels indicate approximate zero values of the gradient. The black and white pixels of mask graphs show absence and presence of photonic component material substance (glass) in its medium (air).

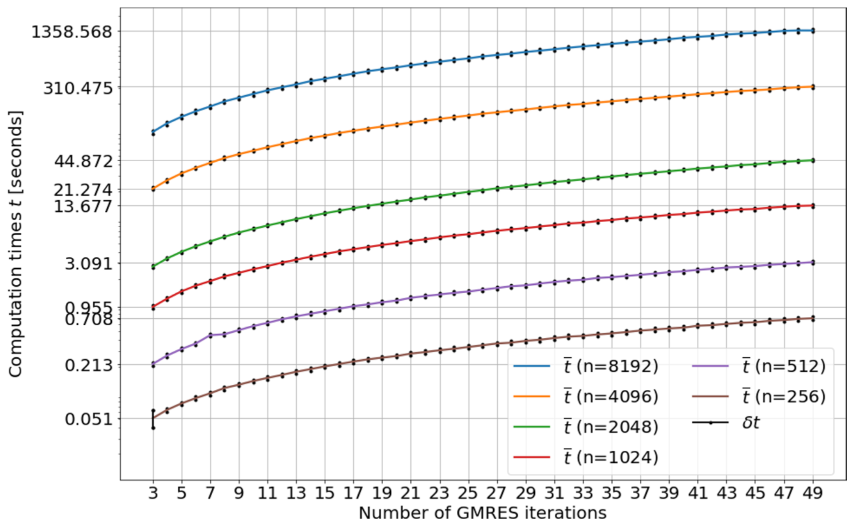
The values of topology optimization target functions are computed as solutions to Helmholtz equation linear system (for 1024×1024 uniform grid). The target function values increase monotonically (with TO iteration) as expected. Topology optimization is based on the discrete gradient matrix computation processed as the linear system (derived in the previous work [28]) computed on the GMRES implementation.

The presented topology optimization method is based on the steepest descent numerical method which adds substance (e.g., glass) in the neighborhood to the edge of the device where the gradient is maximal and positive, while it is removing the substance from device pixels where the gradient is minimal and negative.

5. Results

Residuals and Convergence. Figure 3 (left) shows GMRES residuals of the solution to the linear system given by FFT-GFIEM implemented in *CUDA* C++. The numerical experiments were conducted by using 100 program runs. The different curves are drawn for characteristic matrix sizes $n = 256;\ 512;\ 1024;\ 2048;\ 4096;\ 8192$. The accuracy of the residuals is better than 10%, according to the variation between the 100 program runs. The absolute value of the residual taken at GMRES iteration 31 varies between 10^{-5} and 10^{-2}. It increases with the matrix size n while in general the accuracy of the solution is connected to the accuracy of the discretization scheme $\mathcal{O}\left(\frac{1}{n}\right)$.

Relative Errors and Convergence. The figure 3 (right) shows the *CUDA* C++ FFT-GFIEM solution errors relative to the analytical solution versus number of GMRES iterations n_G for the chosen set of n and number of program runs. Their standard deviations δe are $\leq 10^{-5}$ that is negligible compared to their mean values $\overline{e}$. All curves are flattened after 23 GMRES iterations. In addition, the values of flat parts approximately double with doubled n and equal to 0.238; 0.117; 0.061; 0.034; 0.021; 0.016 respectively. With characteristic matrix size n, measurement precisions increases. That fact is connected to the decrease in error values on device boundaries on the uniform grid.



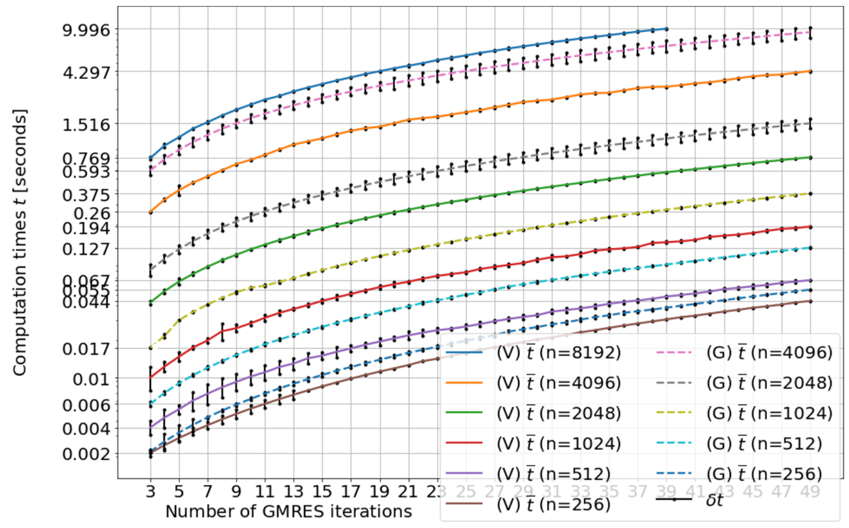


Fig. 4. *MKL* (top) and *CUDA* (bottom) GMRES computational times.

Computational Times. The figures 4 show computational time of C++ GMRES versus the number of GMRES iterations. The variation of the run-time on the CPU between the 100 program runs gives accuracy 1% on these measurements. The GPU program runs are at least 10 times less time-consuming than the CPU ones (using the single-threaded MKL) under the same resolution. The GPU curves are marked with the GPU type: “(G)” for GTX 1080 Ti and “(V)” for Tesla V100 measurements.

Speedups. Figure 5 represents mean values of speedup $\bar{s}$ versus number of GMRES iterations for GTX 1080 Ti and V100 GPUs. The speedup is decreasing with iteration number due to the relative increase of computations multiplication computations of matrices of sizes within $(m+1)\times(m+1)$, where $m \ll n$. The measurement error $\dfrac{\delta s}{\bar{s}}$ is observed to vary between 6% and 24%.

Speedup for the V100 GPU is approximately by a factor of 2 larger as compared to the GTX 1080Ti for each matrix size n. This comparison is shown in Fig. 6. A local minimum for both curves is 2048. That is caused by relatively small computational time growth in MKL version and relatively big computational time growth in *CUDA* version with increase n from 1024 point.

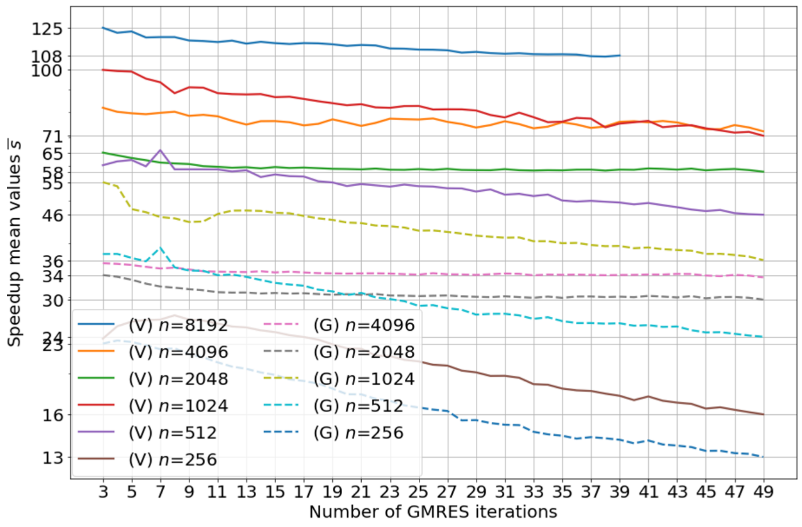


Fig. 5. *CUDA* GMRES speedup means: "(G)" mark means GTX 1080 Ti results and "(V)" mark means Tesla V100 results.

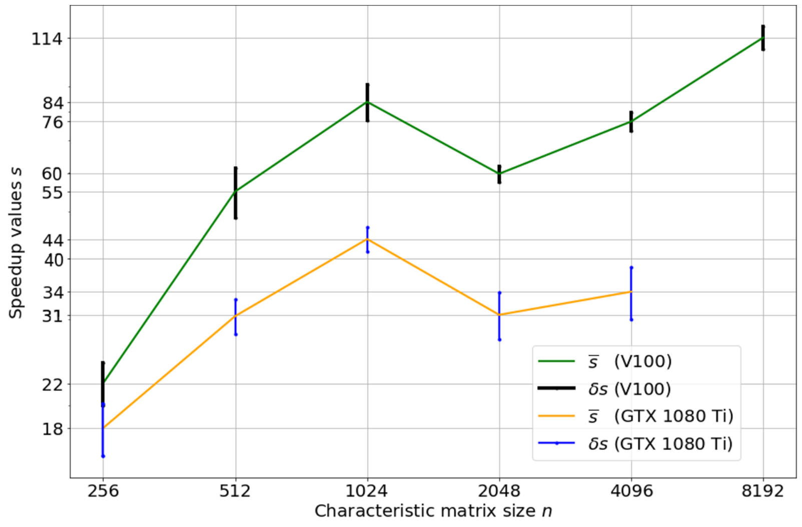


Fig. 6. *CUDA* GMRES speedup vs characteristic matrix size.

6. Conclusion

Our *CUDA* C++ implementation shew the maximal speedup of 55 ($\bar{t} = 0.017$ s) and 125 ($\bar{t} = 0.77$ s) for 1024×1024 (GTX 1080 Ti) and 8192×8192 (V100) dense Toeplitz matrices generated from GFIEM (Helmholtz equation solver) We compared *CUDA* C++ GMRES implementation to the sequential version of this program represented by single-threaded *MKL* C++ GMRES implementation launched on Xeon Gold 6140 CPU. The obtained speedups are bigger than 32 (number of cores of Xeon Gold 6140 CPU). Therefore, the presented GMRES GPU implementation is faster than what can be theoretically possible on the CPU alone. Furthermore, the software can run simultaneous computations of different linear systems by managing separated streams on a GPU (and GPUs).

Linear system matrix size for, e.g., $2^{13} \times 2^{13}$ (i.e., $n = 2^{26}$) uniform grid was reduced from $2^{26} \times 2^{26}$ to $(2^{14} - 1) \times (2^{14} - 1)$ in the Toeplitz matrix GMRES implementation that does not store the whole Toeplitz-like (e.g., Hankel, Circulant) matrix.

Topology optimization target function values are monotonically increasing (as expected) for 1024×1024 uniform grid. That provides the achieved resolution $(170.7$ nodes per wavelength) and computed accuracy (6.1%) of Toeplitz matrix GMRES implementation. We tested this accuracy is acceptable for gradient and target function computations for topology optimization of photonic components.

The average accuracy in the presented application of GMRES implementation to Helmholtz equation linear system can be increased by using mesh refinement on the boundaries because the lowest accuracy is noticed to be on the boundary as is usually expected for a uniform grid. However, the non-uniform grid does not usually provide GMRES implementation with Toeplitz matrix that increases computational time as there is no capability to use matvec FFT-acceleration.

For future studies, the proposed software have opportunities for photonic component fabrication and can be tested by taking into account real usage conditions (noises, elastic vibrations). In addition, that is groundwork for conducting comparison with other software in terms of computational benchmarks, accuracy, topology optimization opportunity (freedom) widths.

Acknowledgment

In our work, we exploited “Zhores” supercomputer installed at Skolkovo Institute of Science and Technology [49]. We also would like to thank Oleg Panarin, Yuri Shkandybin, Anton Maliutin, and Andrey Bykov for technical help and management of “Zhores” cluster.

The work was carried out within the framework of the state task 0030-2019-0008 (“Space”).

REFERENCES

1. T. Søderåard, *Green’s Function Integral Equation Methods in Nano-optics*, CRC Press, Boca Raton (2019).
2. E. Chu and A. George, *Inside the FFT Black Box: Serial and Parallel Fast Fourier Transform Algorithms*, CRC Press, Boca Raton (1999).
3. J. JáJá, *An Introduction to Parallel Algorithms*, vol. 17, Addison-Wesley Reading, New York (1992).
4. A. E. Martínez-Castro, J. A. Molina-Moya, and P. Ortiz, “An iterative parallel solver in gpu applied to frequency domain linear water wave problems by the boundary element method,” *Front. Built Env.*, **4**, 69 (2018).
5. X. Liu, Z. Liu, S. X.-Tan, and A. J. Gordon, “Full-chip thermal analysis of 3D ICs with liquid cooling by GPU-accelerated GMRES method,” in: *Thirteenth International Symposium on Quality Electronic Design (ISQED)* (2012), pp. 123–128; https://doi.org/10.1109/ISQED.2012.6187484.
6. Z. Chen, H. Liu, S. Yu, B. Hsieh, and L. Shao, “Reservoir simulation on nvidia tesla gpus,” *Rec. Adv. Sci. Comp. Appl.*, **586**, 125 (2013).
7. R. Li, and Y. Saad, “GPU-accelerated preconditioned iterative linear solvers,” *J. Supercomp.*, **63**(2), 443–466 (2013); https://doi.org/10.1007/s11227-012-0825-3.
8. I. Yamazaki, H. Anzt, S. Tomov, M. Hoemmen, and J. Dongarra, “Improving the performance of ca-gmres on multicores with multiple gpus,” in: *2014 IEEE 28th International Parallel and Distributed Processing Symposium* (2014), pp. 382–391.
9. R. Couturier, “Designing scientific applications on GPUs,” *Chapman & Hall/CRC Numerical Analysis and Scientific Computing Series*, CRC Press, Boca Raton (2013); https://books.google.ru/books?id=C1 SBQAAQBAJ.
10. G. Marchuk and Y. Kuznetsov, “On the question of optimal iteration processes [in Russian],” in: *Doklady Akademii SSSR*, **181**, 1331–1334 (1968).
11. Y. Saad and M. H. Schultz, “GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems,” *SIAM J. Sc. Stat. Comp.*, **7**(3), 856–869 (1986).
12. I. Dravins, “Numerical implementations of the generalized minimal residual method (GMRES),” *MSc Theses in Math. Sci.* (2015).
13. H. F. Walker and P. Ni, “Anderson acceleration for fixed-point iterations,” *SIAM J. Num. Anal.*, **49**(4), 1715–1735 (2011).
14. J. Drkošová, A. Greenbaum, M. Rozložník, and Z. Strakoš, “Numerical stability of GMRES,” *BIT Num. Math.*, **35**(3), 309–330 (1995).
15. R. Karlson, *A Study of Some Roundoff Effects of the GMRES-Method*, Universitetet i Linköping/Tekniska Högskolan i Linköping, Linköping (1991).
16. G. Meurant, *Computer Solution of Large Linear Systems*, Vol. 28, Elsevier, Amsterdam (1999).
17. Y. T. Feng, D. Peri, and D. R. J. Owen, “A multi-grid enhanced gmres algorithm for elasto-plastic problems,” *Int. J. Num. Meth. Eng.*, **42**(8), 1441–1462 (1998).

18. P. Ghysels, T. Ashby, K. Meerbergen, and W. Vanroose, "Hiding global communication latency in the gmres algorithm on massively parallel machines," *SIAM J. Sci. Comp.*, **35**(1), 48–71 (2013); https://doi.org/10.1137/12086563X; https: //doi.org/10.1137/ 12086563X.
19. C. Vuik, R. R. P. van Nooyen, and A. P. Wesseling, "Parallelism in ILU-preconditioned GMRES," *Par. Comp.*, **24**(14), 1927–1946 (1998); https://doi.org/10.1016/S0167-8191(98)00084-2.
20. M. Harris, "An efficient matrix transpose in CUDA C/C++," *Nvidia*, **26**, 2018 (2013).
21. E. de Sturler, "A parallel variant of GMRES (m)," in: *Proceedings of the 13th IMACS World Congress on Computational and Applied Mathematics*, IMACS, Criterion Press, vol. 9 (1991).
22. M. Bobrov, R. Melton, S. Radziszowski, and M. Lukowiak, "Effects of GPU and CPU loads on performance of CUDA applications," in: *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, p. 1, WorldComp (2011).
23. T. J. Ashby, P. Ghysels, W. Heirman, and W. Vanroose, "The impact of global communication latency at extreme scales on Krylov methods," in: *International Conference on Algorithms and Architectures for Parallel Processing*, Springer (2012), pp. 428–442.
24. E. C. Carson, *Communication-avoiding Krylov subspace methods in theory and practice*, PhD Thesis, UC Berkeley (2015).
25. M. Hoemmen, *Communication-avoiding Krylov subspace methods*, PhD Thesis, UC Berkeley (2010).
26. G. Li, "A block variant of the gmres method on massively parallel processors," *Par. Comp.*, **23**(8), 1005–1019 (1997); https://doi.org/ 10.1016/S0167-8191(97)00004-5.
27. Y. Liu, S. Mukherjee, N. Nishimura, M. Schanz, W. Ye, A. Sutradhar, E. Pan, N. Dumont, A. Frangi, and A. Saez, "Recent advances and emerging applications of the boundary element method," *Appl. Mech. Rev.*, **64**(3), 030802 (2011).
28. I. B. Minin, E. E. Nuzhin, A. I. Boyko, M. S. Litsarev, and I. V. Oseledets, "Evolutionary structural optimization al- gorithm based on fft-jvie solver for inverse design of wave devices," in: *2018 Engineering and Telecommunication (EnT-MIPT)* (2018), pp. 146–150.
29. D. Guide, "Cuda c best practices guide," *NVIDIA*, July (2013).
30. J. Sanders and E. Kandrot, *CUDA by Example: an Introduction to General-Purpose GPU Programming* (2005).
31. M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, et al., "An overview of the Trilinos project," *ACM TOMS*, **31**(3), 397–423 (2005).
32. S. Balay, S. Abhyankar, M. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. Gropp, et al., *Petsc Users Manual* (2019).
33. H. Anzt, T. Cojean, G. Flegar, F. Gbel, T. Grtzmacher, P. Nayak, T. Ribizel, Y. M. Tsai, A. E. S. Quintana-Ortí, Ginkgo, *A Modern Linear Operator Algebra Framework for High Performance Computing* (2020).
34. L. Reichel and Q. Ye, "Breakdown-free gmres for singular systems," *SIAM J. Math. Anal. Appl.*, **26**(4), 1001–1021 (2005).
35. D. Calvetti, B. Lewis, and L. Reichel, "Gmres-type methods for inconsistent systems," *Lin. Alg. Appl.*, **316**(1-3), 157–169 (2000).
36. J. R. Partington, J. R. Partington, et al., *An Introduction to Hankel Operators*, Vol. 13, Cambridge University Press, Cambridge (1988).
37. V. Y. Pan, *Structured Matrices and Polynomials: Unified Superfast Algorithms*, Springer, Boston (2012).
38. S. Kavitha, V. Vijay, and A. Saketh, "Matrix sort-a parallelizable sorting algorithm," *Int. J. Comp. Appl.*, **143**(9), 1–6 (2016).
39. V. Olshevsky, I. Oseledets, and E. Tyrtyshnikov, "Tensor properties of multilevel toeplitz and related matrices," *Lin. Alg. Appl.*, **412**(1), 1–21 (2006).
40. I. B. Minin, pycuGMRES (2020); https://github.com/iurii-minin/pycuGMRES; https://pypi.org/project/pycuGMRES/.
41. I. B. Minin, cuGMRES (2020); https://github.com/iurii-minin/cuGMRES.
42. M. Lucia, F. Maggio, and G. Rodriguez, "Numerical solution of the helmholtz equation in an infinite strip by wiener- hopf factorization," *Num. Meth. Part. Diff. Eq.*, **26**(6), 1247–1274 (2010).
43. R. Borghi, F. Gori, M. Santarsiero, F. Frezza, and G. Schettini, "Plane-wave scattering by a set of perfectly conducting circular cylinders in the presence of a plane surface," *JOSA A*, **13**(12), 2441–2452 (1996).
44. M. P. Bendsoe and O. Sigmund, *Topology Optimization: Theory, Methods, and Applications*, Springer, Berlin (2013).
45. J. Bannister, L. Fratta, and M. Gerla, "Optimal topologies for the wavelength-division optical network," in: *Proc. EFOC/LAN'90*, Munich, Germany (1990), pp. 53–57.
46. S. Banerjee and B. Mukherjee, "The photonic ring: Algorithms for optimized node arrangements," *Fib. & Int. Opt.*, **12**(2), 133–171 (1993).
47. J. Smajic, C. Hafner, and D. Erni, "Optimization of photonic crystal structures," *JOSA A*, **21**(11), 2223–2232 (2004).
48. T. Asano and S. Noda, "Iterative optimization of photonic crystal nanocavity designs by using deep neural networks," *Nanoph.*, **8**(12), 2243–2256 (2019).
49. I. Zacharov, R. Arslanov, M. Gunin, D. Stefonishin, A. Bykov, S. Pavlov, O. Panarin, A. Maliutin, S. Rykovanov, and M. Fedorov, *"Zhores" — Petaflops supercomputer for data-driven modeling, machine learning and artificial intelligence installed in Skolkovo Institute of Science and Technology*, vol. 9, pp. 512–520 (2019); https://doi.org/ 10.1515/eng-2019-0059; https://www.degruyter.com/ view/j/eng.2019.9.issue-1/eng-2019-0059/eng-2019-0059.xml.