Scalable and Fast Characteristic Mode Analysis using GPUs

Khulud Alsultan[1,2], Mohamed Z. M. Hamdalla[1], Sumitra Dey[1], Praveen Rao[3], and Ahmed M. Hassan[1]

[1]Department of Computer Science Electrical Engineering, University of Missouri-Kansas City, Kansas City, MO 64110, USA

[2]Department of Computer Science and Engineering, King Saud University, Riyadh 11451, Saudi Arabia

[3]University of Missouri-Columbia, Columbia, MO 65211, USA

***Abstract* –** Characteristic mode analysis (CMA) is used in the design and analysis of a wide range of electromagnetic devices such as antennas and nanostructures. The implementation of CMA involves the evaluation of a large method of moments (MoM) complex impedance matrix at every frequency. In this work, we use different open-source software for the GPU acceleration of the CMA. This open-source software comprises a wide range of computer science numerical and machine learning libraries not typically used for electromagnetic applications. Specifically, this paper shows how these different Python-based libraries can optimize the computational time of the matrix operations that compose the CMA algorithm. Based on our computational experiments and optimizations, we propose an approach using a GPU platform that is able to achieve up to 16× and 26× speedup for the CMA processing of a single 15k × 15k MoM matrix of a perfect electric conductor scatterer and a single 30k × 30k MoM matrix of a dielectric scatterer, respectively. In addition to improving the processing speed of CMA, our approach provided the same accuracy as independent CMA simulations. The speedup, efficiency, and accuracy of our CMA implementation will enable the analysis of electromagnetic systems much larger than what was previously possible at a fraction of the computational time.

***Index Terms* –** Big data applications, characteristic mode analysis, graphics processing unit, method of moments, scalability.

I. INTRODUCTION

The theory of characteristic modes (TCM), also termed characteristic mode analysis (CMA), is a computational technique that is used in a wide range of electromagnetic applications such as antenna design [1–7], electromagnetic compatibility [8–13], and nano-antenna analysis and design [14–17]. The numerical recipe of the CMA implementation involves the numerical analysis of the method of moments (MoM) impedance matrix using operations such as the singular value decomposition (SVD), multiplication, inverse, slicing, and matrix transpose [18]. CMA of electrically large scatterers or multi-scale scatterers with fine details is challenging since it can generate large MoM impedance matrices that can cause out-of-memory issues, limited resource errors, or longer time to execute [19]. Moreover, if an application requires the CMA of hundreds of frequencies to accurately quantify the electromagnetic response over a wide frequency range, terabytes (TBs) of RAM and storage and high-speed processors are needed since CMA typically involves the processing of one dense matrix per frequency. Therefore, CMA creates a *classical big data problem* that needs advanced computer science and Big Data tools to address efficiently.

A wide range of Big Data tools has recently been developed to accelerate matrix operations in different applications. For example, Lee and Cichocki [20] proposed algorithms for calculating SVD on large-scale matrices based on low-rank tensor train decomposition. Their approach outperformed MATLAB and LOBPCG (locally optimal block preconditioned conjugate gradient). Gu *et al.* designed the Marlin library, which includes three matrix multiplication algorithms to improve the efficiency of large matrix computations [21]. Liu and Ansari used Apache Spark for processing matrix inversions to reduce the computation and space complexity of large-scale matrices [22]. They developed a scalable lower−upper decomposition-based block-recursive algorithm called SparkInverse, which outperformed MRInverse [23] and MPInverse [24] on large matrices (e.g., 102,400 × 102,400 matrices). Yu *et al.* developed MatFast, a scalable matrix processing for in-memory distributed cluster on Apache Hadoop [25] and Spark for large-scale matrices [26]. MatFast supports matrix transpose and multiplication. Recently,

Submitted On: June 29, 2021
Accepted On: January 1, 2022

https://doi.org/10.13052/2022.ACES.J.370203

Misra *et al.* developed Stark, which is a distributed matrix multiplication algorithm using Apache Spark for large-scale matrices [27]. Stark is based on Strassen's matrix multiplication scheme, which is faster than the standard one. It was tested on matrices of size up to 16,384 $\times$ 16,384. While Stark was faster than Marlin and Spark MLlib [28], it has high space complexity.

In this work, motivated by the computational complexity of CMA [41], we used some of the new open-source software for GPU computing previously mentioned, as well as different Python-based numerical libraries, to accelerate the CMA of large-scale matrices. The performance of Python-based numerical libraries was recently reported for simple matrix operations and in an angle of arrival calculation example [42]. However, to the best of our knowledge, these tools are not commonly used in computational electromagnetic applications, and the novelty in this work is to explore their efficacy in accelerating the CMA implementation. We start our optimization by decomposing the CMA algorithm into basic matrix operations. We perform exhaustive computational experiments to study the optimum numerical library to execute each matrix operation and explore whether each operation is better executed on multi-core CPUs or on a GPU. By allocating the operations accordingly, the acceleration of the CMA can be maximized.

It is important to emphasize that, in this work, we do not use electromagnetic concepts such as the multilevel fast multi-pole approximation (MLFMA) [19] or the symmetry and Toeplitz properties of the MoM matrix for arrays [43] to accelerate the CMA implementation. Moreover, there are alternative electromagnetic decompositions that reduce the computational time and improve the accuracy of the CMA [44]. However, in this work, we develop an alternative approach to accelerate the CMA that is based on brute force computer science techniques. To the best of our knowledge, this work is the first time that the acceleration of the CMA implementation was performed using open-source software for GPU computing. Related work had been recently reported for the acceleration of other electromagnetic techniques such as the MoM and the MLFMA. Yang *et al*. accelerated the MLFMA for more than 10 billion unknowns using 2560 processors and more than 30 TB of RAM [45]. However, in this work, we limit our focus to GPU-based acceleration techniques. To put our CMA acceleration work into context, Table 1 summarizes some of this recent work classified by the electromagnetic method that is accelerated, the maximum size of the impedance matrix considered, the acceleration technique, and the speedup achieved. It is important to emphasize that, for conciseness, we limit Table 1 to the studies that used GPUs to accelerate the frequency-domain MoM and other closely related techniques. Therefore, Table 1 does not include acceleration studies that did not report the use of GPUs or studies that used GPUs to accelerate computational electromagnetic techniques that are not related to the MoM. GPU acceleration of the MoM implementation is also available in commercial solvers such as WIPL-D [46].

The rest of this paper is organized as follows. We begin with an overview of CMA in Section II. In Section III, we present our GPU implementation for CMA using different Python numerical libraries. We present the results, including the computational time and validation of the numerical results, followed by a discussion in Section IV. Section V concludes the paper.

II. OVERVIEW OF CMA

CMA decomposes the total surface current generated on a scatterer into a set of fundamental real and orthogonal modes and calculates the relative importance of each mode at each frequency [47]. The modes can be calculated by solving the eigenvalue problem given by

$$\mathbf{X}\mathbf{J}_n = \lambda_n \mathbf{R}\mathbf{J}_n, \tag{1}$$

where $\mathbf{X}$ and $\mathbf{R}$ are the imaginary and real components of the impedance operator $\mathbf{Z}$ [18]. The vectors $\mathbf{J}n$ are the eigenvectors or the eigen-currents, and λ_n are the eigenvalues. The resulting eigenvalues and eigenvectors are independent of the excitation. By applying the MoM [48], eqn (1) can be converted into the matrix equation:

$$[\mathbf{X}]\,[\mathbf{J}_n] = \lambda_n\,[\mathbf{R}]\,[\mathbf{J}_n]. \tag{2}$$

Two approaches have been reported to obtain the impedance matrix $\mathbf{Z}$, namely, the volume integral equation (VIE) formulation [18] and the surface integral equation (SIE) formulation [47]. In this work, we adopt the SIE formulation that requires the surface discretization of the scatterer.

CMA is applied extensively for conducting bodies [47, 49]. However, applying CMA for complex shapes, composed of one or more dielectric materials, is still under development [50]. The CMA analysis of dielectric materials requires post-processing of the impedance matrix because the solution includes both electric and magnetic induced currents ($\mathbf{J}$ and $\mathbf{M}$). Applying the Galerkin methods to the integral equation of SIE, it can be expressed into the following equivalent system of matrix equations [51]:

$$\begin{bmatrix} \mathbf{Z}^{\mathrm{EJ}} & \mathbf{Z}^{\mathrm{EM}} \\ \mathbf{Z}^{\mathrm{HJ}} & \mathbf{Z}^{\mathrm{HM}} \end{bmatrix} \begin{bmatrix} \mathbf{J} \\ \mathbf{M} \end{bmatrix} = \begin{bmatrix} \mathbf{V}^{\mathrm{E}} \\ \mathbf{V}^{\mathrm{H}} \end{bmatrix} \tag{3}$$

To solve the equation for only the electric currents, the system of the matrix in eqn (3) can be modified by replacing the magnetic currents in this equation as follows [51]:

$$\mathbf{M} = \left(\mathbf{Z}^{\mathrm{HM}}\right)^{-1}\left(\mathbf{V}^{\mathrm{H}} - \mathbf{Z}^{\mathrm{HJ}}\mathbf{J}\right). \tag{4}$$

Table 1: Review of recent research on the acceleration of various electromagnetic techniques

Ref.	Electromagnetic technique/comments	Maximum size of the impedance matrix	Acceleration technique	Speedup ratio (with respect to single CPU)		
				Impedance matrix assembly	Solution of linear system	Total
[29]	Conventional MoM/ Single precision NVIDIA GeForce 7600GT (675 MHz) GPU with 256 MB video memory and an AMD Athlon 64 3000+ (1.81 GHz) CPU with 1 GB memory	∼ 9.9k	GPU acceleration on Brook platform	17.33	NA	NA
[30]	Conventional MoM/ Complex double-precision A CUDA-capable device, GeForce GTX 280 built on the GT200 architecture was used	∼ 7.7k	GPU CUDA acceleration	∼ 140	∼ 13	∼ 45
[31]	Conventional MoM/ Complex double-precision Intel Core i7, GeForce GTX 275 and the CUDA API, PGI Fortran+CULA, Intel Fortran + MKL were used	∼ 7k	GPU CUDA acceleration	∼ 9	∼ 5	∼ 6
[32]	Conventional MoM/ Complex double-precision NVIDIA GT200 (GeForce GTX 275) CUDA-capable device was used as an external math coprocessor to the host CPU (2.66-GHz Intel Core i7)	∼ 7k	GPU CUDA acceleration	∼ 13	∼ 5	∼ 8.5
[33]	Conventional MoM/ Single precision Intel Core i7 CPU 930 @2.8GHz, 24 GB of RAM, Windows 7 Professional 64-bit. Up to three identical GPUs GeForce GTX 480, 1536 MB of VRAM (each), VRAM access speed 177 GB/s. Four hard-disk drives (HDD) with I/O speed 100 MB/s (per HDD, without buffering).	∼ 152k	Out-of-core solver accelerated with multiple GPUs	NA	NA	20
[34]	Single-level fast multi-pole method (FMM) Complex double-precision Multi-node GPU cluster of 13 nodes, and Nvidia Tesla M2090 GPU per node. An MVAPICH2 implementation of MPI was used for cluster parallel programming.	∼ 245k	13 nodes GPU cluster	NA	NA	∼ 700
[35]	MLFMA Single precision The CPU-MLFMA is parallelized and executed by eight threads on a workstation with a four-core Intel Xeon processor W3550 (with a clock speed of 3.06 GHz). The OpenMP-CUDA-MLFMA is executed on four Nvidia Tesla C2050 GPUs.	∼ 342k	OpenMP-CUDA on multiple GPUs	124	NA	21
[36]	Higher-order MoM (HMoM)/ Complex double-precision A Dell Precision 5400 equipped with two Intel Xeon quad-core CPUs (2.5 GHz clock speed), 16 GB RAM, and one NVIDIA GTX 660 GPU card was used. GPU's architecture is Kepler GK104 with a core frequency of 1015 MHz, 960 Stream Processors (SPs). CUDA version 4.2 was used.	∼ 70k	Optimized parallel out-of-core LU solver on hybrid GPU/CPU platform	6	9.78	NA
[37]	FMM-FFT/ Dual Xeon system with four R9 280X cards	∼ 1100k	GPU/CPU hybrid platform	NA	NA	∼ 30
[38]	MLFMA/ Single precision Intel i7 processor with 8 GB RAM, a TESLA C2075 GPU with 6 GB of RAM, a Windows 7 64-bit license, and the 3.1 CUDA toolkit were used.	∼ 694k	Parallelization using CUDA	189	–	84
[39]	Load-balanced out-of-GPU memory implementation of MoM/ Double-precision Intel quad-core i7 3820 CPU running at 3.6 GHz with64 GB RAM, and a GeForce GTX 680 GPU with 4 GB of on-board memory running at 1006 MHz were used	–	GPU CUDA acceleration	2.21 (when compared to four-core CPU)	NA	NA
[40]	MoM Intel Xeon E5-2698 v3 processor with 16 cores and 25 GB of RAM and an NVIDIA Tesla K40 GPU were used	∼ 1M	FMM/GPU	NA	2.5	NA

Substituting eqn (4) back into eqn (3), it can be expressed as [51]

$$\left[\mathbf{Z}^{\mathrm{EJ}} - \mathbf{Z}^{\mathrm{EM}}\left(\mathbf{Z}^{\mathrm{HM}}\right)^{-1}\mathbf{Z}^{\mathrm{HJ}}\right]\mathbf{J} = \mathbf{V}^{\mathrm{E}} - \mathbf{Z}^{\mathrm{EM}}\left(\mathbf{Z}^{\mathrm{HM}}\right)^{-1}\mathbf{V}^{\mathrm{H}}. \tag{5}$$

From eqn (5), a new effective impedance matrix can be expressed as

$$\mathbf{Z}^{\mathrm{E}} = \mathbf{Z}^{\mathrm{EJ}} - \mathbf{Z}^{\mathrm{EM}}\left(\mathbf{Z}^{\mathrm{HM}}\right)^{-1}\mathbf{Z}^{\mathrm{HJ}}. \tag{6}$$

It is worth noting that the previous equation involves the processing of complex matrices. Most of the conventional Big Data tools can only handle pure-real matrices, especially on GPUs. However, recently, Big Data tools were developed to handle complex matrices on GPU [52]. Therefore, one of the main contributions of this work is to identify, in the following sections, the Big Data tools compatible with complex matrices necessary for the CMA of dielectric scatterers, as shown in eqn (6) [51]. Using the new equivalent impedance matrix shown in eqn (6), a new generalized eigenvalue equation can be formulated by [51]

$$[\mathbf{X}^{\mathrm{E}}][\mathbf{J}_n] = \lambda_n[\mathbf{R}^{\mathrm{E}}][\mathbf{J}_n], \tag{7}$$

where λ_n and $\mathbf{J}n$ are the eigenvalues and eigenvectors calculated using $\mathbf{R}^E$ and $\mathbf{X}^E$, which are the real and imaginary parts, respectively, of the equivalent impedance matrix $\mathbf{Z}^E$. The eigenvalues λ_n can be used to calculate the modal significance MSn of each mode as

$$\mathrm{MS}n = 1/\left|1 + j\lambda_{\mathrm{n}}\right|. \tag{8}$$

The modal significance is independent of the excitation, and it identifies the relative weight of each mode at any given frequency. The modal significance varies between 0 and 1, reaching 1 typically at the resonance frequency of the mode [14]. It is important to emphasize that there are alternative implementations for performing the CMA of dielectric scatterers. Huang *et al*. performed an excellent review and comparison in [53]. However, the goal of this work is to explore GPU-based acceleration, and the techniques developed herein have the potential to yield similar acceleration levels in alternative dielectric CMA implementations.

A scatterer that is highly complex in shape or electrically large needs a detailed mesh that yields a large MoM impedance matrix containing thousands of rows and columns. These matrices consume gigabytes of disk space and RAM for storage during analysis. Computing CMA for hundreds of frequencies needs the analysis of hundreds of large MoM impedance matrices, which also pose a Big Data challenge.

With the availability of high-end CPUs and hardware accelerators such as GPUs, one can cope with the Big Data challenge in CMA. CPU cores and GPUs provide internal parallelism inside their architecture [54]. This can speed up the matrix computations in CMA. A GPU computing platform provides promising support toward improving resource utilization [54]. Today, open-source software such as TensorFlow [55], designed originally for large-scale machine learning, and Python libraries such as NumPy [52] and CuPy [56] can be exploited for CMA.

Thus, a hybrid CPU/GPU platform provides ample opportunities to test different techniques for accelerating different matrix operations using open-source software that can handle large datasets.

Algorithm 1 CMA pseudocode based on Harrington's method [18]

Input: Real part of input MoM matrix $\mathbf{R}$, imaginary part of input MoM matrix $\mathbf{X}$, matrix size n_cols
Result: Eigenvalues λ_n and eigenvectors $\mathbf{J}_n$
Using CPU (TensorFlow):
  read $\mathbf{R}$; read $\mathbf{X}$
Using GPU (CuPy):
  $\mathbf{Z} = \mathbf{R} + \mathbf{X}$ ▷ Create complex matrix
  Slice $\mathbf{Z}$ into four parts $\mathbf{Z}_{EJ}, \mathbf{Z}_{EM}, \mathbf{Z}_{HJ}$, and $\mathbf{Z}_{HM}$
  $\mathbf{ZZ} = \mathbf{Z}_{EJ} - \mathbf{Z}_{EM} * \mathbf{Z}_{HM}^{-1} * \mathbf{Z}_{HJ}$
  $\mathbf{RR}$= real($\mathbf{ZZ}$)
  $\mathbf{XX}$ = imag($\mathbf{ZZ}$)
  $\mathbf{U}, \mathbf{S}, \mathbf{V}$=SVD($\mathbf{RR}$)
  $si = len(\mathbf{S})$
  $u_{11} = \mathbf{S}^{-0.5}$ ▷ $u_{11} = 1/sqrt(S)$
  $\mathbf{A} = \mathbf{U}^{\mathbf{T}} * \mathbf{XX} * \mathbf{U}$
  Slice $\mathbf{A}$ into four parts $\mathbf{A}_{11}, \mathbf{A}_{12}, \mathbf{A}_{21}$, and $\mathbf{A}_{22}$
  $\mathbf{B} = u_{11} * (\mathbf{A}_{11} - \mathbf{A}_{12} * \mathbf{A}_{22}^{-1} * \mathbf{A}_{12}^{T}) * u_{11}$
  $\mathbf{UB}, \mathbf{SB}, \mathbf{HB} = SVD(\mathbf{B})$
  $\mathbf{VB} = \mathbf{U} * concat(eye(si), (-\mathbf{A}_{22}^{-1} * \mathbf{A}_{12}^{T})) * u_{11} * \mathbf{HB}$
  $\mathbf{J}_n$ = flip $\mathbf{VB}$
  for $jm = 1, 2, \ldots, n\_cols$ **do**
    $\lambda_n[jm] = imag(\mathbf{J}_n[:, jm]^T * \mathbf{ZZ} * \mathbf{J}_n[:, jm])$
  end for
return

III. ACCELERATION OF THE CMA IMPLEMENTATION ON A GPU PLATFORM

In this section, we develop multiple different CMA implementations using different hardware setups and different numerical libraries. We then perform extensive experiments to identify the optimum implementation for each matrix size and for each hardware setup. We used three different hardware setups for our CMA implementation: (1) a multi-core CPU, (2) a GPU platform, and (3) a hybrid CPU/GPU platform. We used the following numerical libraries: (1) TensorFlow2.0 (TF), (2) Numpy Python library, and (3) CuPy Python library. TF is an open-source platform for machine learning, and it can be executed on both CPUs and GPUs. On a hybrid CPU/GPU platform, TF will assign all operations to the GPU by default. To instruct TF to execute a certain operation on a CPU, the following statement needs to be added before the operation:

```
with tf.device (device name):
```

TF has application programming interfaces (APIs) in several languages such as C++, Python, and Java. In this work, we used Python to implement CMA with TF. The NumPy python library can run on a multi-core CPU, whereas the CuPy python library can only run on GPUs. Therefore, we developed five different CMA evaluations as follows: (1) TF where all matrix operations are executed only on CPUs, (2) TF where all matrix operations are executed only on GPUs, (3) hybrid TF implementation where some matrix operations are executed on CPUs and some matrix operations are executed on GPUs, (4) NumPy where all matrix operations are executed only on CPUs, and (5) CuPy where all matrix operations are executed only on GPUs. The goal is to identify the fastest implementation out of the five for different matrix sizes. Moreover, the five implementations will guide future CMA users who have access to only CPUs or GPUs and will also guide users who prefer to use one of the previously described Python libraries.

Our CMA implementation is based on the method described in Algorithm 1 [18]. We chose this particular implementation since it is capable of accurately handling a wide range of scatterers, including wires and wire-like nanostructures [14]. First, we read the real and imaginary parts of the input MoM matrix using TensorFlow (Lines 3−4). Next, we construct the complex matrix **Z** followed by slicing it into four equal parts and then computing **ZZ** (Lines 5−10). This step is only performed for dielectric targets following the approach in [51]. For PEC scatterers, this step is skipped, and **ZZ** is set equal to **Z**. The SVD process is then performed (Lines 10 and 11). After that, matrices **A** and **B** are computed as detailed in Lines 13−16. The remaining steps are to compute the eigenvalues λ_n as shown in Lines 17−22.

Algorithm 2 Mode tracking pseudocode

Input: Real part of $\mathbf{Z}$ matrix $\mathbf{RR}$, eigenvectors at the previous frequency $\mathbf{oldM}$, no. of requested modes $\mathbf{numM}$
Result: Ordered eigenvalues $\mathbf{CM}$
for $im = 1, 2, \ldots, \mathbf{numM}$ **do**
  for $jm = 1, 2, \ldots, \mathbf{numM}$ **do**
    $\mathbf{CM}[im, jm] = abs(\mathbf{oldM}[:, im]^T * \mathbf{RR} * \mathbf{J}_n[:, jm])$
  end for
  $[temp, It] = max(\mathbf{CM}[im, :])$
  if $It \neq im$ **then**
    $temp1 = \lambda_n[It]$
    $\lambda_n[It] = \lambda_n[im]$
    $\lambda_n[im] = temp1$
    $temp2 = \mathbf{J}_n[:, It]$
    $\mathbf{J}_n[:, It] = \mathbf{J}_n[:, im]$
    $\mathbf{J}_n[:, im] = temp2$
  end if
end for
return

The modes and eigenvalues generated by the CMA (Algorithm 1) are not ordered in the same way over the entire frequency range [57–59]. Mode tracking is, therefore, performed to find the correct mode ordering throughout the frequency range of interest. Our implementation of mode tracking, which can be run on a CPU or a GPU, is based on calculating the correlation between the modes of the current frequency and the modes of the previous frequency [57] (see Algorithm 2).

IV. EXPERIMENTAL SETUP, RESULTS, AND DISCUSSION

In this section, we report the performance of the five CMA implementations previously described. We ran all experiments on CloudLab [60], an experimental testbed for cloud computing. We used a machine in CloudLab's Wisconsin data center with two Intel Xeon E5-2667 8-core CPUs (3.20 GHz) and an NVIDIA Tesla V100 SMX2 GPU (16 GB). All the algorithms were implemented and evaluated using the following software and tools: Linux Ubuntu 16.04, TensorFlow 2.0.0, CUDA 10.0.130, Python 3.7.10., NumPy1.20.1, CuPy 8.3.0, and Pandas 1.2.3.

Table 2 breaks down the computational time for the different CMA matrix operations for a 14k $\times$ 14k matrix using the five implementations previously described: TF on CPU, TF on GPU, TF on hybrid CPU/GPU, NumPy on CPU, and CuPy on GPU. All CPU computational experiments in Table 2 used 32 cores. Comparing the computational time for the TF on CPU and TF on GPU in Table 2, we see that TF on GPU is faster than TF on CPU for all matrix operations except for the SVD and the writing of the eigenvector operation. Therefore, to optimize the TF on a hybrid CPU/GPU platform, we assigned all CMA matrix operations to GPU except the SVD and the writing of the eigenvectors operation, which were assigned to the CPU. Table 2 shows that the TF on hybrid CPU/GPU is faster than the TF on CPU or TF on GPU.

The fourth implementation, NumPy on CPU, is faster than the three TF implementations in Table 2. The main advantage of the NumPy on CPU is its acceleration of the matrix multiplications and the SVD, even though it is slower than TF in terms of the matrix inverse operation and writing the eigenvectors. Finally, the CuPy on GPU is the fastest implementation with a significant acceleration in the matrix multiplication and the SVD compared to the other four implementations. The CuPy on GPU can provide a speedup of 80$\times$ compared to other implementations in Table 2, highlighting the importance of selecting the optimum numerical library for the CMA implementation. The analysis in Table 2 shows that different numerical libraries generate drastic differences in the computation time of different matrix

Table 2: Time distribution (minutes) for 14k $\times$ 14k dielectric object over TF CPU, TF-GPU, hybrid TF, NumPy with multi-core CPU, and CuPy with GPU implementation

Matrix operation	TF CPU	TF GPU	TF Hybrid	NumPy with multi-core CPU	CuPy with GPU
Reading the matrix	0.33	0.25	0.28	0.20	0.22
Assembling real and imaginary parts	0.25	0.03	0.03	0.10	0.08
Multiplications	151.07	6.33	6.38	1.22	0.25
Inverse	3	0.03	0.02	0.17	0.05
SVDs	9.78	28.23	9.75	0.92	0.35
Writing eigenvectors	0.27	0.92	0.23	0.93	0.90
Total time	164.83	38.7	16.79	3.45	1.91

Table 3: MOM matrix memory requirements for different formats

Matrix type	Matrix size	CSV file size	Binary file size
4k $\times$ 4k	4776 $\times$ 4776	1.5 GB	350 MB
14k $\times$ 14k	14,183 $\times$ 14,183	27 GB	3.0 GB
15k $\times$ 15k	15,279 $\times$ 15,279	32.4 GB	3.6 GB
16k $\times$ 16k	16,608 $\times$ 16,608	36 GB	4.2 GB
30k $\times$ 30k	33,024 $\times$ 33,024	138 GB	16.2 GB

operations, which, to the best of our knowledge, was not documented for large dense MoM matrices processed by CMA. If GPUs are not available, the NumPy implementation provides the fastest implementation of CMA, whereas if GPUs are available, the CuPy implementation is the fastest. Therefore, Table 2 can be used as a guide for choosing the optimum numerical library for any computational electromagnetic technique based on the dominant matrix operations of its algorithm.

To quantify the scalability of the CMA implementation, we tested MoM impedance matrices, of different sizes, generated by the commercial electromagnetic solver FEKO [61]. Matrices of both dielectric and PEC matrices were tested. Details of these matrices, including the matrix size, the size of the CSV file, and the binary file size, are shown in Table 3. We used the binary files storing the MoM matrices for these experiments. The advantage of the binary format is that it requires approximately 10%$-$20% of the storage hard drive memory required by the ASCII and CSV file formats, as shown in Table 3. This reduction in storage memory is particularly important for the CMA of large MoM matrices and/or for the simulation of many matrices to cover multiple frequencies.

Tables 4 and 5 show the computational time required by our implementation for dielectric and PEC scatterers, respectively. We tested the computational time of the

Table 4: Time taken by our CMA implementations on a multi-core CPU and GPU (minutes) for dielectric

Matrix type	No. of cores used on the multi-core CPU (NumPy)					GPU (CuPy)
	1	4	8	16	32	
4k $\times$ 4k	0.88	0.44	0.36	0.34	0.34	0.29
14k $\times$ 14k	20.44	6.74	4.38	3.48	3.48	1.90
16k $\times$ 16k	33.69	10.81	6.86	5.23	5.26	2.83
30k $\times$ 30k	257.13	79.43	47.14	45.11	32.29	15.60

Table 5: Time taken by our CMA implementations on a multi-core CPU and GPU (minutes) for PEC

Matrix type	No. of cores used on the multi-core CPU (NumPy)					GPU (CuPy)
	1	4	8	16	32	
15k $\times$ 15k	128.98	38.25	23.82	17.92	18.12	12.38

CMA NumPy implementation using 1, 2, 4, 8, 16, and 32 cores without GPU, and we also added the computational time required when only a GPU and the CuPy implementation were employed. As we increased the number of cores, the computational time decreased. For instance, it took 257 minutes to process the 30k $\times$ 30k matrix on 1 core but only 32 minutes on 32 cores. For a PEC scatterer, represented by a 15k $\times$ 15k matrix, it took 130 minutes to process the matrix on 1 core but only 18 minutes on 16 cores. Tables 4 and 5 show that moving from 16 cores to 32 cores showed no decrease in the computational time for matrix sizes of 16k $\times$ 16k and smaller. Therefore, for matrices that are 16k $\times$ 16k and smaller, the maximum speedup is achieved at 16 cores. However, Table 4 shows that, for the 30k $\times$ 30k matrix, increasing the number of cores from 16 to 32 lowered the computational time and enhanced the speedup, indicating the potential of our implementation to scale for matrices 30k $\times$ 30k and larger.

In Tables 4 and 5, we also report the computational time required by our CMA CuPy implementation on a GPU. While our implementation required around 32.29 minutes to process a 30k $\times$ 30k matrix of a dielectric scatterer using a 32-CPU cores, it took only 15.6 minutes on the GPU platform. We also tested our CMA implementation for a PEC scatterer represented by a 15k $\times$ 15k matrix. Again, the CMA implementation on a GPU platform was the fastest, as shown in Table 5. Using a GPU achieved a speedup of 16$\times$ and 10$\times$ for the dielectric and the PEC object, respectively, in comparison to a single CPU core. The computational time and speedup are shown in Figure 1. Moreover, if we do not consider the time needed to write the eigenvectors in the speedup calculations, the speedup will be 26$\times$ and 16$\times$ for the dielectric and the PEC object, as shown in Figure 2.

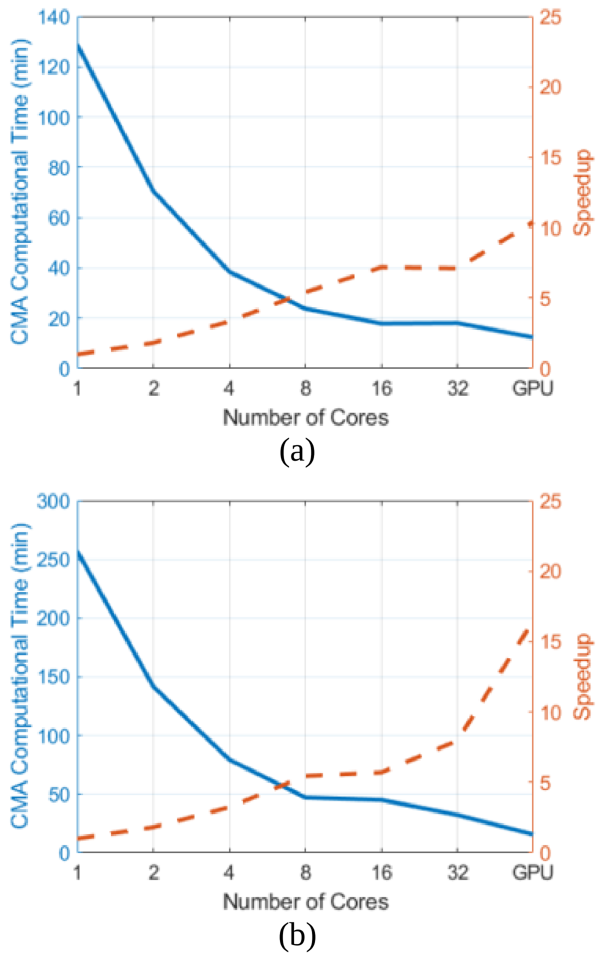


Fig. 1. Speedup and time taken for CMA with writing the eigenvalues and eigenvectors vs. different number of cores for (a) PEC (15k × 15k) and (b) dielectric object (30k × 30k).

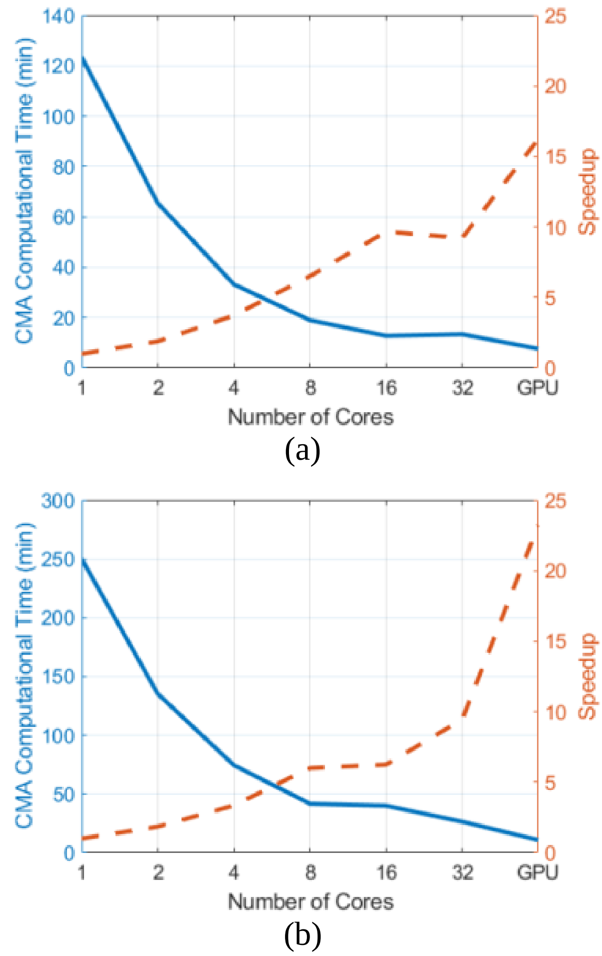


Fig. 2. Speedup and time taken for CMA without writing the eigenvalues and eigenvectors vs. different number of cores for (a) PEC (15k × 15k) and (b) dielectric object (30k × 30k).

Lastly, we validated the numerical results produced by our CMA implementation to demonstrate that it does not compromise accuracy. We tested our implementation for two different cases. For PEC scatterers, we used the horn antenna in [see Figure 3(a)]. The eigenvalues λ_n of the horn antenna calculated using our implementation perfectly match the eigenvalues calculated using FEKO, as shown in Figure 4. We also used a lossless dielectric cylinder of radius 5.25 mm, height 4.6 mm, ε_r = 38, and μ_r = 1 [see Figure 3(b)]. The frequency range was chosen from 4.5 to 7.5 GHz with a 50-MHz interval. This case is often used to verify the results of CMA formulations for real materials [53]. Figure 5 presents the modal significance of the dielectric cylinder, which matches with the results reported by Chen *et al.* [62]. The previous two cases demonstrate the validity of our accelerated CMA implementation.

In this work, we limited our computational experiments to common Big Data tools such as TensorFlow, Python-based CuPy, and Python-based NumPy. Many additional algorithms have been previously reported for speeding matrix operations [63, 64]. In future work, we plan to investigate these implementations and

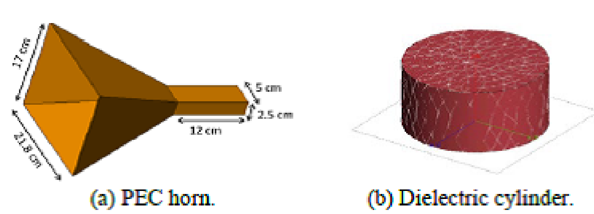


Fig. 3. Different scatterers used to validate the accuracy of our accelerated CMA implementation. (a) PEC horn antenna. (b) Dielectric cylinder.

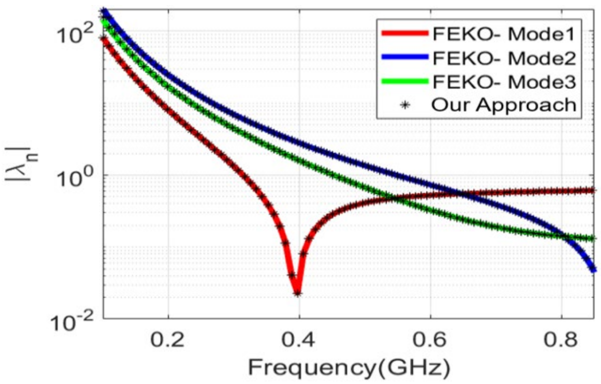


Fig. 4. Eigenvalues for a horn-shaped PEC.

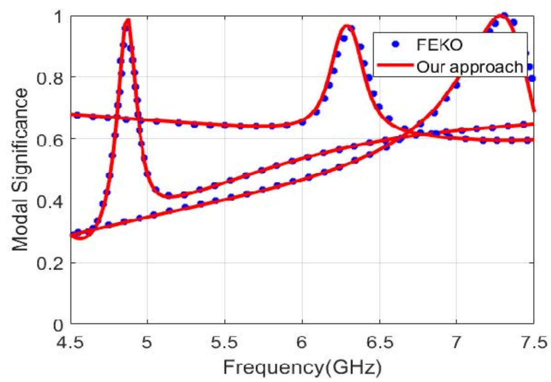


Fig. 5. Modal significance for a cylinder-shaped dielectric.

other alternatives, for further acceleration of the CMA implementation.

V. CONCLUSION

In this paper, we tested different numerical implementations of the CMA algorithm using open-source software for GPU computing. We showed that different numerical implementations can have drastically different computational times for the different matrix operations that make up the CMA algorithm. Therefore, it is important to select the optimum numerical library since the computational time can vary for large matrices by up to approximately two orders of magnitude. From our computational experiments, we showed that the CuPy implementation on GPU delivered the largest speedup. In comparison to the execution on a single CPU core, the CuPy implementation on GPU was capable of achieving 26× and 16× speedup for processing a single MoM matrix of a dielectric and a PEC object, respectively. In addition to faster execution, our implementation provided the same accuracy as theoretical solutions and independent commercial CMA simulations.

ACKNOWLEDGMENT

Khulud Alsultan would like to acknowledge the support of the Saudi Arabian Cultural Mission. This work was supported in part by Office of Naval Research (ONR) grants: #N00014-17-1-2932 and #N00014-17-1-3016. “Distribution Statement A. – Approved for public release. Distribution is unlimited.”

REFERENCES

[1] M. Cabedo-Fabres, E. Antonino-Daviu, A. Valero-Nogueira, and M. F. Bataller, “The theory of characteristic modes revisited: A contribution to the design of antennas for modern applications,” *IEEE Antennas and Propagation Magazine*, vol. 49, no. 5, pp. 52–68, 2007.

[2] M. Vogel, G. Gampala, D. Ludick, U. Jakobus, and C. Reddy, “Characteristic mode analysis: Putting physics back into simulation,” *IEEE Antennas and Propagation Magazine*, vol. 57, no. 2, pp. 307–317, 2015.

[3] G. Angiulli, G. Amendola, and G. Di Massa, “Application of characteristic modes to the analysis of scattering from microstrip antennas,” *Journal of Electromagnetic Waves and Applications*, vol. 14, no. 8, pp. 1063–1081, 2000.

[4] L. Guan, Z. He, D. Ding, and R. Chen, “Efficient characteristic mode analysis for radiation problems of antenna arrays,” *IEEE Transactions on Antennas and Propagation*, vol. 67, no. 1, pp. 199–206, 2019.

[5] N. Michishita and H. Morishita, “Helmet antenna design using characteristic mode analysis,” *Applied Computational Electromagnetics Society (ACES) Journal*, vol. 35, no. 2, p. 6, 2020.

[6] M. M. Elsewe and D. Chatterjee, “Ultra-wide bandwidth enhancement of single-layer single-feed patch antenna using the theory of characteristic modes,” *Applied Computational Electromagnetics Society (ACES) Journal*, vol. 33, no. 3, p. 4, 2018.

[7] A. Araghi and G. Dadashzadeh, “Detail-oriented design of a dual-mode antenna with orthogonal radiation patterns utilizing theory of characteristic modes,” *Applied Computational Electromagnetics Society (ACES) Journal*, vol. 28, no. 10, p. 8, 2013.

[8] Q. Wu, H.-D. Bruns, and C. Schuster, “Characteristic mode analysis of radiating structures in digital systems,” *IEEE Electromagnetic Compatibility Magazine*, vol. 5, no. 4, pp. 56–63, 2016.

[9] X. Yang, Y. S. Cao, X. Wang, L. Zhang, S. He, H. Zhao, J. Hu, L. Jiang, A. Ruehli, J. Fan, and J. L. Drewniak, “EMI radiation mitigation for heatsinks using characteristic mode analysis,” *in 2018 IEEE Symposium on Electromagnetic Compatibility, Signal Integrity and Power Integrity (EMC, SI & PI)*. IEEE, pp. 374–378, 2018.

[10] S. H. Yeung and C.-F. Wang, "Exploration of characteristic mode theory for electromagnetic compatibility modeling," *in 2018 IEEE International Symposium on Electromagnetic Compatibility and 2018 IEEE Asia-Pacific Symposium on Electromagnetic Compatibility (EMC/APEMC)*. IEEE, pp. 1278–1282, 2018.

[11] M. Z. Hamdalla, B. Bissen, A. N. Caruso, and A. M. Hassan, "Experimental validations of characteristic mode analysis predictions using GTEM measurements," *in 2020 IEEE International Symposium on Antennas and Propagation and USNC-URSI Radio Science Meeting*. IEEE, 2020.

[12] M. Z. Hamdalla, A. M. Hassan, and A. N. Caruso, "Characteristic mode analysis of the effect of the UAV frame material on coupling and interference," *in 2019 IEEE International Symposium on Antennas and Propagation and USNC-URSI Radio Science Meeting*. IEEE, pp. 1497–1498, 2019.

[13] M. Z. Hamdalla, A. M. Hassan, A. Caruso, J. D. Hunter, Y. Liu, V. Khilkevich, and D. G. Beetner, "Electromagnetic interference of unmanned aerial vehicles: A characteristic mode analysis approach," *in 2019 IEEE International Symposium on Antennas and Propagation and USNC-URSI Radio Science Meeting*. IEEE, pp. 553–554, 2019.

[14] A. M. Hassan, F. Vargas-Lara, J. F. Douglas, and E. J. Garboczi, "Electromagnetic resonances of individual single-walled carbon nanotubes with realistic shapes: A characteristic modes approach," *IEEE Transactions on Antennas and Propagation*, vol. 64, no. 7, pp. 2743–2757, 2016.

[15] P. Ylä-Oijala, D. C. Tzarouchis, E. Raninen, and A. Sihvola, "Characteristic mode analysis of plasmonic nanoantennas," *IEEE Transactions on Antennas and Propagation*, vol. 65, no. 5, pp. 2165–2172, 2017.

[16] K. C. Durbhakula, A. M. Hassan, F. Vargas-Lara, D. Chatterjee, M. Gaffar, J. F. Douglas, and E. J. Garboczi, "Electromagnetic scattering from individual crumpled graphene flakes: a characteristic modes approach," *IEEE Transactions on Antennas and Propagation*, vol. 65, no. 11, pp. 6035–6047, 2017.

[17] S. Dey, D. Chatterjee, E. J. Garboczi, and A. M. Hassan, "Plasmonic nanoantenna optimization using characteristic mode analysis," *IEEE Transactions on Antennas and Propagation*, vol. 68, no. 1, pp. 43–53, 2019.

[18] R. Harrington and J. Mautz, "Computation of characteristic modes for conducting bodies," *IEEE Transactions on Antennas and Propagation*, vol. 19, no. 5, pp. 629–639, 1971.

[19] Q. I. Dai, J. Wu, H. Gan, Q. S. Liu, W. C. Chew, and E. Wei, "Large-scale characteristic mode analysis with fast multipole algorithms," *IEEE Transactions on Antennas and Propagation*, vol. 64, no. 7, pp. 2608–2616, 2016.

[20] N. Lee and A. Cichocki, "Big data matrix singular value decomposition based on low-rank tensor train decomposition," *in International Symposium on Neural Networks*. Springer, pp. 121–130, 2014.

[21] R. Gu, Y. Tang, Z. Wang, S. Wang, X. Yin, C. Yuan, and Y. Huang, "Efficient large scale distributed matrix computation with Spark," *in 2015 IEEE International Conference on Big Data (Big Data)*. IEEE, pp. 2327–2336, 2015.

[22] J. Liu, Y. Liang, and N. Ansari, "Spark-based large-scale matrix inversion for big data processing," *IEEE Access*, vol. 4, pp. 2166– 2176, 2016.

[23] J. Xiang, H. Meng, and A. Aboulnaga, "Scalable matrix inversion using mapreduce," *in Proceedings of the 23rd International Symposium on High-Performance Parallel and Distributed Computing*, pp. 177–190, 2014.

[24] L. S. Blackford, J. Choi, A. Cleary, E. D'Azeuedo, J. Demmel, I. Dhillon, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, R. C. Whaley, and J. J. Dongarra, ScaLAPACK User's Guide. USA: Society for Industrial and Applied Mathematics, 1997.

[25] T. White, Hadoop: The Definitive Guide, 1st ed. O'Reilly Media, Inc., 2009.

[26] Y. Yu, M. Tang, W. G. Aref, Q. M. Malluhi, M. M. Abbas, and M. Ouzzani, "In-memory distributed matrix computation processing and optimization," *in 2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. IEEE, pp. 1047–1058, 2017.

[27] C. Misra, S. Bhattacharya, and S. K. Ghosh, "Stark: Fast and scalable Strassen's matrix multiplication using Apache Spark," IEEE Transactions on Big Data, 2020.

[28] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, D. Xin, X. Reynold, M. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar, "Mllib: machine learning in apache spark," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1235–1241, 2016.

[29] S. Peng and Z. Nie, "Acceleration of the method of moments calculations by using graphics processing units," *IEEE Transactions on Antennas and Propagation*, vol. 56, no. 7, pp. 2130–2133, 2008.

[30] E. Lezar and D. B. Davidson, "GPU-accelerated method of moments by example: Monostatic

scattering," *IEEE Antennas and Propagation Magazine*, vol. 52, no. 6, pp. 120–135, 2010.

[31] T. Topa, A. Karwowski, and A. Noga, "Using GPU with CUDA to accelerate MoM-based electromagnetic simulation of wiregrid models," *IEEE Antennas and Wireless Propagation Letters*, vol. 10, pp. 342–345, 2011.

[32] T. Topa, A. Noga, and A. Karwowski, "Adapting MoM with RWG basis functions to GPU technology using CUDA," *IEEE Antennas and Wireless Propagation Letters*, vol. 10, pp. 480–483, 2011.

[33] D. P. Zoric, D. I. Olcan, and B. M. Kolundzija, "Solving electrically large EM problems by using out-of-core solver accelerated with multiple graphical processing units," *in 2011 IEEE International Symposium on Antennas and Propagation (APSURSI)*. IEEE, pp. 1–4, 2011.

[34] Q. M. Nguyen, V. Dang, O. Kilic, and E. El-Araby, "Parallelizing fast multipole method for large-scale electromagnetic problems using GPU clusters," *IEEE Antennas and Wireless Propagation Letters*, vol. 12, pp. 868–871, 2013.

[35] J. Guan, S. Yan, and J.-M. Jin, "An openMP-CUDA implementation of multilevel fast multipole algorithm for electromagnetic simulation on multi-GPU computing systems," *IEEE Transactions on Antennas and Propagation*, vol. 61, no. 7, pp. 3607–3616, 2013.

[36] X. Mu, H.-X. Zhou, K. Chen, and W. Hong, "Higher order method of moments with a parallel out-of-core LU solver on GPU/CPU platform," *IEEE Transactions on Antennas and Propagation*, vol. 62, no. 11, pp. 5634–5646, 2014.

[37] M. J. Miranda, T. Ö zdemir, and R. J. Burkholder, "Hardware acceleration of an FMM-FFT solver using consumer-grade GPUs," *in 2016 United States National Committee of URSI National Radio Science Meeting (USNC-URSI NRSM)*. IEEE, pp. 1–2, 2016.

[38] E. Garćıa, C. Delgado, L. Lozano, and F. Ćatedra, "Efficient strategy for parallelisation of multilevel fast multipole algorithm using CUDA," *IET Microwaves, Antennas & Propagation*, vol. 13, no. 10, pp. 1554–1563, 2019.

[39] T. Topa, "Load-balanced fortran-based out-of-GPU memory implementation of the method of moments," *IEEE Antennas and Wireless Propagation Letters*, vol. 16, pp. 813–816, 2017.

[40] R. Adelman, N. A. Gumerov, and R. Duraiswami, "FMM/GPUaccelerated boundary element method for computational magnetics and electrostatics," *IEEE Transactions on Magnetics*, vol. 53, no. 12, pp. 1–11, 2017.

[41] K. Alsultan, P. Rao, A. Caruso, and A. Hassan, "Scalable Characteristic Mode Analysis: Requirements and Challenges," *in Large Scale Networking (LSN) Workshop on Huge Data: A Computing, Networking and Distributed Systems Perspective*, pp. 1–2, 2020.

[42] A. Weiss and A. Elsherbeni, "Computational performance of matlab and python for electromagnetic applications." *Applied Computational Electromagnetics Society (ACES) Journal*, vol. 35, no. 11, 2020.

[43] S. Kaffash, R. Faraji-Dana, M. Shahabadi, and S. Safavi-Naeini, "A fast computational method for characteristic modes and eigenvalues of array antennas," *IEEE Transactions on Antennas and Propagation*, 2020.

[44] D. Tayli, M. Capek, L. Akrou, V. Losenicky, L. Jelinek, and M. Gustafsson, "Accurate and efficient evaluation of characteristic modes," *IEEE Transactions on Antennas and Propagation*, vol. 66, no. 12, pp. 7066–7075, 2018.

[45] M.-L. Yang, B.-Y. Wu, H.-W. Gao, and X.-Q. Sheng, "A ternary parallelization approach of mlfma for solving electromagnetic scattering problems with over 10 billion unknowns," *IEEE Transactions on Antennas and Propagation*, vol. 67, no. 11, pp. 6965– 6978, 2019.

[46] B. L. Mrdakovic, M. M. Kostic, D. I. Olcan, and B. M. Kolundzija, "New generation of wipl-d in-core multi-gpu solver," *in 2018 IEEE International Symposium on Antennas and Propagation & USNC/URSI National Radio Science Meeting*. IEEE, pp. 413–414, 2018.

[47] Y. Chang and R. Harrington, "A surface formulation for characteristic modes of material bodies," *IEEE Transactions on Antennas and Propagation*, vol. 25, no. 6, pp. 789–795, 1977.

[48] R. F. Harrington, "Matrix methods for field problems," Proceedings of the IEEE, vol. 55, no. 2, pp. 136–149, 1967.

[49] M. Khan and D. Chatterjee, "Characteristic mode analysis of a class of empirical design techniques for probe-fed, U-slot microstrip patch antennas," *IEEE Transactions on Antennas and Propagation*, vol. 64, no. 7, pp. 2758–2770, 2016.

[50] Q. Wu, "Characteristic mode assisted design of dielectric resonator antennas with feedings," *IEEE Transactions on Antennas and Propagation*, vol. 67, no. 8, pp. 5294–5304, 2019.

[51] Y. Chen and C.-F. Wang, "Surface integral equation based characteristic mode formulation for dielectric resonators," *in 2014 IEEE Antennas and Propagation Society International Symposium (APSURSI)*. IEEE, pp. 846–847, 2014.

[52] T. E. Oliphant, A guide to NumPy. Trelgol Publishing USA, vol. 1, 2006.

[53] S. Huang, J. Pan, C.-F. Wang, Y. Luo, and D. Yang, "Unified implementation and cross-validation of the integral equationbased formulations for the characteristic modes of dielectric bodies," *IEEE Access*, vol. 8, pp. 5655–5666, 2019.

[54] S. Mittal and J. S. Vetter, "A survey of CPU-GPU heterogeneous computing techniques," *ACM Computing Surveys (CSUR)*, vol. 47, no. 4, pp. 1–35, 2015.

[55] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," arXiv preprint arXiv:1603.04467, 2016.

[56] R. Nishino and S. H. C. Loomis, "Cupy: A numpy-compatible library for NVIDIA GPU calculations," *in Proceedings of Workshop on Machine Learning Systems (LearningSys) in the Thirty first Annual Conference on Neural Information Processing Systems (NIPS)*, 2017.

[57] M. Capek, P. Hazdra, P. Hamouz, and J. Eichler, "A method for tracking characteristic numbers and vectors," *Progress In Electromagnetics Research*, vol. 33, pp. 115–134, 2011.

[58] J. Zhu, W. Li, and L. Zhang, "Broadband Tracking of Characteristic Modes," *Applied Computational Electromagnetics Society (ACES) Journal*, vol. 34, no. 11, p. 6, 2019.

[59] Q. He, Z. Gong, H. Ke, and L. Guan, "A Double Modal Parameter Tracking Method To Characteristic Modes Analysis," *Applied Computational Electromagnetics Society (ACES) Journal*, vol. 32, no. 12, p. 8, 2017.

[60] D. Duplyakin, R. Ricci, A. Maricq, G. Wong, J. Duerig, E. Eide, L. Stoller, M. Hibler, D. Johnson, K. Webb, A. Akella, K. Wang, G. Ricart, L. Landweber, C. Elliott, M. Zink, E. Cecchet, S. Kar, and P. Mishra, "The design and operation of CloudLab," in 2019 USENIX Annual Technical Conference (USENIX ATC 19). Renton, WA: USENIX Association, pp. 1–14, Jul. 2019. [Online]. Available: https://www.usenix.org/conference/atc19/presentation/duplyakin

[61] 2021. [Online]. Available: https://www.altair.com/feko/

[62] Y. Chen, "Alternative surface integral equation-based characteristic mode analysis of dielectric resonator antennas," *IET Microwaves, Antennas & Propagation*, vol. 10, no. 2, pp. 193– 201, 2016.

[63] R. Solca, A. Haidar, S. Tomov, T. C. Schulthess, and J. Dongarra, "Poster: A novel hybrid cpu-gpu generalized eigensolver for electronic structure calculations based on fine grained memory aware tasks," *in 2012 SC Companion: High Performance Computing, Networking Storage and Analysis*. IEEE, pp. 1340–1340, 2012.

[64] D. Liu, R. Li, D. J. Lilja, and W. Xiao, "A divide-and-conquer approach for solving singular value decomposition on a heterogeneous system," *in Proceedings of the ACM International Conference on Computing Frontiers*, pp. 1–10, 2013.

Khulud Alsultan received the B.S. degree in computer science from King Saud University, Riyadh, Saudi Arabia in 2006, the M.Sc. degree in computer science from Kent State University, Kent, OH, USA, in 2013, and the Ph.D. degree in computer science and telecommunication and computer networking from the University of Missouri-Kansas City, Kansas City, MO, USA, in 2021.

She is currently a Lecturer with the Department of Computer Science, King Saud University (KSU). Her research interest includes Big Data and data analysis, and its applications in healthcare and electrical engineering. She served as an external reviewer for several conferences, such as *BEXA 2019* and *BDA 2019*. While being an active researcher, she obtained the Preparing Future Faculty program (PFF) graduate certificate at UMKC. Moreover, Dr. Alsultan received awards including the Grace Hopper Celebration Student Scholarship in 2018, and the Graduate Student Travel Grant, UMKC, Kansas City, MO, USA, in 2019.

Mohamed Hamdalla received the B.Sc. and M.Sc. degrees from the Arab Academy for Science, Technology and Maritime Transport, Alexandria, Egypt, in 2012 and 2016, respectively, both in electronics and communications engineering, and the Ph.D. degree in electrical engineering from the University of Missouri-Kansas City, Kansas City, MO, USA, in 2021.

His current research interests include antennas, metamaterials, microwave filters, electromagnetic compatibility and interference, characteristic mode theory, and applications.

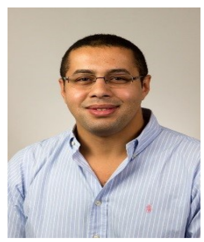
Sumitra Dey received the B.Tech. degree in radio physics and electronics from the University of Calcutta, Kolkata, India, in 2014, the M.Tech. degree in RF and microwave communication engineering from IIEST Shibpur, West Bengal, India, in 2016, and the Ph.D. degree in electrical engineering from the University of Missouri-Kansas City, Kansas City, MO, USA, in 2021.

She is currently a Lead Product Engineer with Multiphysics System Analysis Group, Cadence Design Systems, San Jose, CA, USA. Her research interests include computational electromagnetics, signal/power integrity in high frequency, nano-electromagnetics, nondestructive evaluation, experimental microwave and terahertz imaging, AI/ML based optimization of electromagnetic response, multilayer Green's functions, characteristic mode theory, and applications.

Dr. Dey was awarded the Honorable Mention in Student paper competition in *2020 Applied Computational Electromagnetic Society (ACES) Conference*, Monterey, CA, USA, Honorable Mention in Student paper competition in *2019 IEEE APS/USNC URSI Symposium*, Atlanta, GA, USA, Honorable Mention in 2018 Altair FEKO Student Design Competition, and the Best Student Paper in *IEEE CALCON 2015*, Kolkata, India.

Praveen Rao received the M.S. and Ph.D. degrees in computer science from the University of Arizona, Tucson, AZ, USA, in 2007 and 2001, respectively.

He is currently a tenured Associate Professor with joint appointment in the Department of Health Management & Informatics and the Department of Electrical Engineering & Computer Science at the University of Missouri (MU), Columbia, MO, USA. His research interests are in the areas of Big Data management, data science, health informatics, and cybersecurity. He directs the Scalable Data Science (SDS) Lab at MU. His research, teaching, and outreach activities have been supported by the National Science Foundation (NSF), Air Force Research Lab (AFRL), the National Endowment for the Humanities (NEH), the National Institutes of Health (NIH), the University of Missouri System (Tier 1 grant, Tier 3 grant), University of Missouri Research Board, and companies. He is a Co-PI for the NSF IUCRC Center for Big Learning. At MU, he is a core faculty of the Center for Biomedical Informatics (CBMI), the Cybersecurity Center, the MU Institute for Data Science and Informatics, and the CERI Center. He is a core scientist of the Washington University Center for Diabetes Translation Research funded by NIH. He is a Senior Member of the ACM (2020) and IEEE (2015).



Ahmed M. Hassan received the B.Sc. (with highest honors) and M.Sc. degrees from Cairo University, Giza, Egypt, in 2004 and 2006, respectively, both in electronics and communications engineering. He received the Ph.D. degree in electrical engineering from the University of Arkansas, Fayetteville, AR, USA, in 2010.

From 2011 to 2012, he was a Postdoctoral Researcher with the Department of Electrical Engineering, University of Arkansas. From 2012 to 2015, he was a Postdoctoral Researcher with the National Institute of Standards and Technology, Gaithersburg, MD, USA. He is currently an Assistant Professor with the Computer Science Electrical Engineering Department, University of Missouri-Kansas City. His current research interests include nanoelectromagnetics, bioelectromagnetics, electromagnetic compatibility and interference, nondestructive evaluation, experimental microwave, and terahertz imaging.