Parallel GPU Optimization of the Shooting and Bouncing Ray Tracing Methodology for Propagation Modeling

Stephen Kasdorf, *Student Member, IEEE,* Blake Troksa, *Student Member, IEEE,* Cam Key, *Student Member, IEEE,* Jake Harmon, *Member, IEEE,* Sudeep Pasricha, *Fellow, IEEE,* and Branislav M. Notaroš, *Fellow, IEEE*

***Abstract*—We propose a novel unified parallelization framework, consisting of algorithms, strategies, and data structures, to radically enhance the efficiency of the shooting and bouncing rays (SBR) method for ray tracing (RT) electromagnetic propagation modeling. The massively parallel optimization of the SBR code is achieved by integration of the SBR with NVIDIA OptiX Prime programming interfaces on graphics processing units (GPUs), comprehensive parallelization of all components of the SBR algorithm, including electric field computation and postprocessing tasks being traditionally limited to sequential operation, and addressing and optimizing memory usage and constraints to further advance efficiency of the overall method. Numerical results demonstrate that the new proposed optimized SBR methodology achieves massive parallel vs. serial speedups and upwards of 99% parallelism under Amdahl's parallelization scaling law. The strategic use of GPUs and the innovative meticulous parallel optimization of all computational aspects of the code, explained in detail in the paper, yield an SBR ray tracing methodology of unparalleled efficiency, without sacrificing the previously advanced and established accuracy of the method. Rather, the presented major enhancements of the efficiency and the uniquely high level of parallelism are enabled by and addressed synergistically with the improvements of the accuracy of the SBR computation.**

***Index Terms* – Wireless propagation modeling, asymptotic high-frequency techniques, ray tracing, shooting bouncing rays, high performance computing, parallelization, graphics processing units, parallel/serial speedup, parallelism level, scaling.**

I. Introduction

Ray tracing (RT) is a high-frequency asymptotic method that provides an alternative approach to solving electrically large computational electromagnetics (CEM) problems that traditionally require prohibitively large computation times using full-wave CEM solvers [1]-[3]. As a significant advantage in terms of efficiency over full-wave methods, the shooting and bouncing rays (SBR) RT technique features a time complexity of $O(NK)$, where N is the number of rays and K is the number of reflections [3], [4]. The SBR algorithm also has substantially lower computational complexity than the image theory (IT), another RT approach, the complexity of which is $O(N^K)$, where N is the number of planar facets in the model and K is the number of reflections. IT is also limited to planar structures because in curved geometries the concept of images becomes ambiguous with an image from a curved surface being a line instead of a discrete point [5].

Generally, CEM background and advancements tightly rely on those in (1) electromagnetic/physical formulations and contexts of engineering problems, (2) mathematical and numerical foundations of methods and algorithms, and (3) computing hardware and software infrastructure. Consequently, CEM research is one of the most natural, intense, and advanced combinations of engineering, physics, mathematics, and computer science, with extremely exciting potential and challenges. Indeed, it is the category (3) that has recently provided unprecedented opportunities, with the associated challenges, for the rapid growth of CEM capabilities and modeling technologies needed and desired by electromagnetics application researchers and practitioners, in close synergy, however, with categories (1) and (2). This paper focusses on some of the new ways to better harness the advantages of emerging and growing computing hardware and software infrastructure for CEM, and RT in particular.

Most notably, the development and abundance of graphics processing units (GPUs) has created the enabling technology to achieve massive parallelization. In fact, enormous advances in GPU technology have made these hardware accelerators widely useful outside of the computer graphics world. Originally for graphics applications, GPU architectures are optimized for inherently parallel operation. GPUs are optimized to handle tens of thousands of threads to keep up with the billions of linear algebra calculations required for a high-end video game rendering in real time on an HD monitor. GPUs are therefore ideal candidates when large scale parallelization of simple operations is required [6]. Due to their unique architecture, GPU programming is often done with the compute unified device architecture (CUDA)

This work was supported by the National Science Foundation under grant ECCS-1646562.

Stephen Kasdorf, Blake Troksa, Cam Key, Jake Harmon, Sudeep Pasricha, and Branislav M. Notaroš are with the Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO 80523-1373 USA (e-mail: skasdorf@rams.colostate.edu, blake.troksa@gmail.com, camkey@rams.colostate.edu, jake.harmon@ieee.org, sudeep.pasricha@colostate.edu, notaros@colostate.edu).

language, an application programming interface (API) created by GPU manufacturer NVIDIA for use with their products.

In addition to its advantages in terms of memory and computation time requirements, the SBR RT method can benefit immensely from GPU acceleration, due to the linear nature of the ray decomposition, enabling independent tracing of each ray (excluding ray overlap processing). Considerable work has been done to parallelize the method for implementation on GPUs, such as [7], [8]. However, with the SBR RT application consisting of (1) geometric and physical path calculation of rays as they propagate through a given structure and (2) field calculation of each ray, it is part (1) where good parallelism has been achieved. Portions of the code remain serial, particularly part (2), namely, electric field computations from RT results.

Accuracy improvements of several components of the SBR computation can lead to a SBR method capable of performing wireless propagation modeling of tunnel environments with the same accuracy as the IT RT, a computationally much slower but traditionally more accurate solver [3], [9]. A ray classification method presented in [9] allows for parallelization of the double count removal portion of the SBR algorithm, which traditionally has been very difficult to parallelize.

This present paper proposes a novel unified parallelization framework of algorithms, strategies, and data structures to radically enhance the efficiency of the SBR RT method by integration of SBR with OptiX Prime, comprehensive parallelization of all components of SBR computation, including electric field computation and postprocessing tasks being traditionally limited to sequential operation, and addressing and optimizing memory usage and constraint issues to further advance efficiency of the overall method. The paper shows excellent parallelism of the method, as well as extremely substantial reductions in computation time when compared to the existing methods.

Details on the approach to computing, storing, and updating the electric field for each ray are described in order to develop an understanding of the flow of data and processing on the GPU. We introduce the important optimization strategies used for writing efficient executable code on NVIDIA GPUs. This includes the importance of structured global memory accesses that limit the total number of reads and writes to memory necessary and the inevitability of GPU synchronization. Configuration optimization of the GPU kernels for the RT program is compared with changes in performance from different configurations. Specific implementations of the RT algorithm to incorporate these strategies are discussed. We believe that the presented algorithms, strategies, and data structures should be very useful to researchers, developers, and practitioners of ray-based and other high-frequency methods and modeling. They should also be of significant interest to the area of parallelism and efficiency enhancement of CEM techniques in general.

The results demonstrate that the new SBR methodology achieves upwards of 99% parallelism, whereas no such scaling performance, or close to it, has been reported for RT computations so far. We assess the level of parallelism using Amdahl's law, a predominant scaling law for systematic prediction, assessment, and understanding of parallelization speedup and parallel scaling properties of a code [6]. This dramatically reduces computation time for convergence in tunnels (tunnels are challenging for RT simulations due to the difficulty in simulating multipath propagation effects in large tunnel environments by RT approaches), for example, as well as in an urban outdoor wireless propagation environment. Overall, the computational accelerations achieved from the very high level of parallelism yield an SBR ray tracing methodology of unparalleled efficiency, without sacrificing the previously advanced and established accuracy of the method [3]. Importantly, the major enhancements of the efficiency (this present work) have been enabled by and addressed synergistically with the improvements of the accuracy [3] of the SBR computation. Note that a portion of this work has been reported in a M.S. Thesis [10]. A very preliminary brief report has also been presented in [11]. Finally, we insist here on a distinction between parallelization and parallel optimization of the code; namely, without the enhancements proposed in this paper, conventional parallelized SBR will result in several orders of magnitude more computation time for the same accuracy.

II. OptiX Prime Integration in RT CEM Analysis

In our approach to a RT CEM analysis of a propagation structure or system, the environment is meshed into discrete triangular facets. We use the NVIDIA OptiX application programming interface (API) to compute the ray-facet intersections in order to trace the rays as they propagate through the environment once launched by the antenna. In most scenarios, the medium of propagation is assumed to be air. Once the ray has reflected, the triangles or faces that have been intersected are used to calculate Fresnel coefficients, which are then used to calculate the attenuation of the electric field based on the material composition of the facet. Such generated ray information is used to populate a buffer of ray type objects, and this buffer is one of the main programming components available in the NVIDIA OptiX Prime RT API. The buffer is passed to the OptiX Prime ray intersection program that runs on the GPU and determines the path of a ray from an origin point to a destination point along the direction specified in the spawning. It is indeed advantageous to use the OptiX Prime API given a tremendous research and implementation effort by NVIDIA over the years in developing OptiX Prime as a general-purpose RT engine [12].

An essential feature and a primary reason for using OptiX Prime is the efficient construction of the binary space partition (BSP) tree. The BSP tree is a tree type data structure that stores subdivisions of an environment and allows for efficient ray triangle intersection tests by pruning out sections of the space that are not physically reachable from other sections of the space. In order to construct the BSP-tree and perform ray-facet intersection tests, the OptiX Prime RT API needs a geometry description file to build a mesh. The OptiX Prime API then loads the mesh and executes a closest hit

query for every ray in the ray buffer and calculates the nearest intersection with a triangle for that ray. Here, an efficient BSP tree reduces the total number of facets that have to be checked for a ray in order to find which one it hits first [13]. The field evaluation can then be performed based on the computed intersections and the associated ray reflections. After the ray reflection calculations, the information regarding the previous path segments the ray has traveled is used in postprocessing to determine any intersections the ray may have had with an observation point.

Note that certain anomalous scenarios can occur while a ray is in the geometric portion of the SBR RT algorithm, and they need proper attention and mitigation. The first such scenario occurs when the ray escapes the mesh that it is propagating in, which is a product of numerical precision error that allows rays that hit directly on the boundary of two facets to escape the mesh structure. This happens rarely and is mitigated in the RT application by identifying the escaping rays in the buffer and removing the respective information from the simulation. In the second scenario, a ray leaves the region of interest, or the section where the observation points are located, without leaving the actual mesh. In order to limit this region of interest, a bounding box consisting of multiple "absorbing" planes is created to terminate any ray that intersects the box.

III. Optimization of Parallelized SBR Ray Tracing CEM Methodology on GPUs

A. SBR RT GPU Computation Theory and Implementation

The shooting bouncing RT program involves the spawning, propagation, electric field computation, and electric field summation of millions of rays. The ability to launch millions of rays is highly dependent on the structure and efficiency of the code as well as the level of parallelization achieved. SBR RT is a highly parallel procedure because each ray propagates independently and any computation involving a ray can be performed without interference from other rays. GPUs appear to be ideally suited for an overreaching goal of fully benefiting from the inherent parallelism of the SBR process, in terms of making the computation as efficient as possible. Specifically, GPUs are effective with this type of highly parallel programming as each basic work unit, or thread, on the GPU can compute, track, and update an individual ray's information. In contrast to central processing units (CPUs), GPUs are optimized for data parallel throughput computations and low latency access to cached data sets. Whereas CPUs are designed to control logic for out of order and abstract execution, GPUs perform best on organized instruction sets with limited control-flow branching. GPUs are also architecturally more tolerant of memory latency and have physically more transistors that can be dedicated to arithmetic computation [13]. For all these reasons, GPUs are highly advantageous for RT applications.

The compute unified device architecture (CUDA) parallel computing platform is the primary language used to program on NVIDIA GPUs, permitting the creation of parallel computational instructions to very substantially increase efficiency from the GPU cores and decrease computation time. The device code for NVIDIA GPUs is written in CUDA C/C++ and the executable code on the GPU runs kernels that are synchronous computation batches containing sequential instruction sets. The sequential code located inside the kernels is executed by the threads that are spawned at the start of the kernel execution.

Understanding the structure and execution procedure of NVIDIA GPUs is essential for efficient parallelization of the SBR RT application. The finest execution unit on the GPU is the thread, with GPU threads being similar to threads on a CPU and executing instructions sequentially. They are grouped into thread blocks and all thread blocks used in the execution of a kernel compose a grid. When a kernel is executed on the GPU, the thread blocks in the grid are distributed to the available streaming multiprocessors (SMs) located on the GPU. Thread blocks do not begin execution until an SM has enough resources for all the threads inside the thread block to execute. The SMs have shared memory that can be used by all the threads inside that block. The threads located in the block running on the SM cannot all run at one time and the SM typically, in the case of the NVIDIA GTX 1060 GPU used for the SBR RT application in this work, can execute a group of 32 threads, known as a warp, at one time. Warps run in parallel, allowing these groups of threads to execute their instructions rapidly.

Parallel execution, however, introduces new complications: race conditions are extremely frequent in parallel programs and especially in the RT application and can destabilize a program without conscious handling. The main race condition occurs when the electric field at an observation point is modified by adding the electric field from all the rays that have intersected the same observation point. Since many threads on the GPU are all executing concurrently, many of the threads would be modifying the memory containing the electric field for the observation point simultaneously. This means that the electric field for each observation point may not be updated properly by the threads performing the postprocessing calculations. To eliminate this race condition, the threads must synchronize to ensure mutual exclusion when reading or writing to the memory where the electric field for the observation points is stored. The downfall with this synchronization is a loss in computation time due to threads having to wait for one another to update the electric field.

Effective GPU programing ensures that work is given to all the pipelines available on the GPU and increases computation throughput by hiding the latency of the memory pipelines. To maximize efficiency on the GPU, ideally all the SMs should be running code continuously with other computations running asynchronously. The amount of memory available on the GPU and on the host CPU are the primary limiting factors of the SBR RT application. The memory requirements of the GPU only allow a fixed number of rays to be computed at a time. Because of this constraint, for some larger structures, the rays need to be split into batches before being sent to the

GPU. The first type of batching involves the subdivision of the faces of the icosahedron used to launch the rays initially (we employ an icosahedron for the sampling method in our spawning technique for the RT application) [3]. The other type of batching utilized in the SBR RT application is the batching of the reflection calculations for each of the rays. If the ray's reflection calculations are not batched, then storing all the path information for a ray over hundreds of reflections would overflow the memory. This is because each ray reflection consists of the ray's previous and current intersection points each represented as three floating point numbers and the identification of the triangle that was hit represented by an integer type. This stored information equates to 28 bytes of memory for each reflection, and for 100 million rays, this would be 2.8 gigabytes of storage per reflection. Hence, as the number of reflections increases, only information about a ray's current reflection segment is tracked to limit this memory footprint, and after all paths in a batch of rays have been computed, the next batch is processed. This procedure is repeated until no remaining rays need to be traced for intersections with the observation spheres or the limit on the number of reflections has been reached.

Maximum efficiency with any kind of parallelization, especially on GPUs, requires the optimization of the program to get the best performance possible. The first type of optimization prevalent with all NVIDIA CUDA GPU programs is the launch configurations of the kernel. The launch configurations refer to the number of blocks and number of threads per block that are created to run the code within the kernel. The downside of having too many spawned threads is that some threads will be idle while the other threads are doing all the work. Conversely, if there are too few threads then there could be resources on the GPU such as CUDA cores and SMs that are not being utilized for computation. To get the best efficiency, all compute units on the GPU should remain occupied with computation. The RT application is memory bound, meaning that the bottleneck in simulations is primarily due to the memory limits available to hold data, and the simulation therefore requires higher thread occupancy in order to achieve better efficiency. In our work, various kernel launch configurations were tested beginning with a block size of 128 threads and adjusted to the number of threads per block in increments of 32. The best results, in general, were obtained with 128 threads per block. The number of blocks is also an important consideration, and typically 1,000 or more thread blocks are necessary as this enables the code to be evenly distributed among the GPU hardware, but also highly modular and scalable with additional GPUs.

B. Optimal Parallelization of Ray Tracing Method

Fig. 1 shows the flow between the GPU and CPU in our SBR RT method, depicting the transfers between the GPU and CPU, which can be a severe bottleneck when running simulations. As can be seen, the method starts with loading the environment mesh into the CPU, which has all pertinent information about surface normals and locations. The ray-initialization-icosahedron is then generated, and the ray-neighbor adjacency map is calculated, which, in turn, is used to find exact ray cone sizes for computing ray-receiver intersections more exactly. This is the bulk of the work performed on the CPU, which remains constant as ray numbers and reflection counts increase. The batching loop is then started, with the 20 faces of the icosahedron being the default number of batches in the program. Initial ray directions are generated across one face of the icosahedron, subdivided according to the n^{th} triangular number [3], and these are copied to the GPU memory.

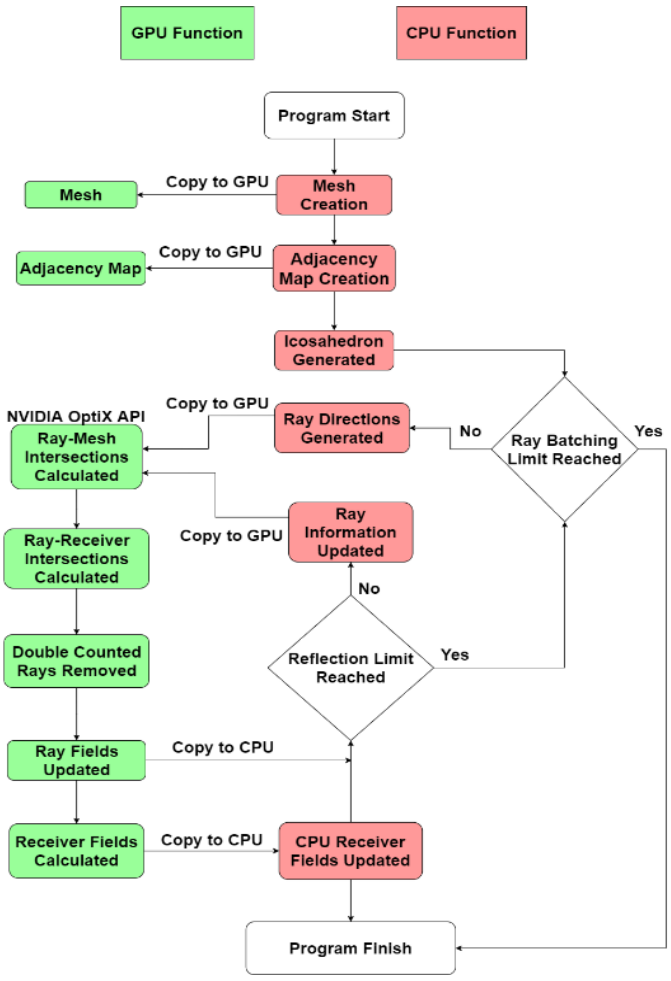

Fig. 1. RT methodology: Flowchart showing basic processes of the proposed RT method as well as transfers between the GPU and CPU.

Ray-mesh intersections are calculated using the NVIDIA OptiX Prime API. In this routine, we determine the intersections of these rays with set receiver points, and remove double counted rays by means of a new method described in [3], [9], which uses information from the ray adjacency map already created. After the ray double count removal, the ray fields are employed to update the receiver fields at every point, according to Fresnel's coefficients and reflection geometry. The ray information and receiver field arrays are copied to the CPU memory, which is the main transfer bottleneck – not severe, however, since both arrays contain a minimal amount of information. The copied receiver

fields are added to the master copy on the CPU. If the reflection limit has not been reached, new ray directions are calculated, and the loop is repeated. If the reflection limit is reached, the program moves onto the next batch of rays, and the process repeats. Once all the batches have been completed, the master CPU copy of receiver fields is output to a text file. As we can see from this process, the bulk of the calculations are done on the GPU, which helps to vastly accelerate the program. Bottlenecks between the GPU and CPU are limited, which also helps to increase speed.

Global memory (memory allocated on the GPU device from the host) throughput is an important optimization consideration when computing on a GPU. Transferring information from host memory to device memory is very slow and the number of copies needs to be minimized. In the RT application, the data for each ray is transferred to the GPU only once before the postprocessing after each reflection. The largest memory storage on the GPU is global memory which is up to 6 GB in the case of NVIDIA GTX 1060 GPU used in this work. Global memory is accessible by all threads running across all thread blocks in the grid, but it has very high access latency [14]. All memory operations performed in the execution of the kernel are issued per warp and the warp accesses memory in 32-word chunks. If the 32 threads in a warp access continuous addresses in memory, then the latency for that group of threads is reduced. These types of accesses to global memory are called coalesced accesses and they ensure all threads access a continuous chunk in memory and therefore reduce the total time for memory access. In order to achieve the coalesced memory accesses, our SBR RT application avoids scattered address patterns and global memory storage with large strides between accesses.

Ray double counting is a well-known issue with the SBR method, with different methods proposed to address this problem [15]-[17]. Recently, we proposed a double count removal method using icosahedron geometry and ray cones approach [3]. As illustrated in Fig. 2, if two neighbor rays reach the same observation point in the same number of reflections, they represent the same wavefront, and one is easily removed. Efficient parallelization of the double count removal procedure is a very important aspect of the GPU optimization for the SBR code. However, the double count removal methods still suffer from the need to examine neighbor ray paths to identify the double counted rays, which is inherently not parallel. To parallelize this portion of the code, adjacent rays need to be split into different classes. The icosahedron spawning pattern allows for grouping rays into three distinct classes, in which no adjacent rays will be a member of the same class. Fig. 3 shows an example of the different classes displayed as three colors plotted over the faces of the icosahedron. Specifically, we split the adjacent rays into separate classes to ensure that no two adjacent rays that double count on the same observation point are processed at the same time, and this classing method is described in detail in [9]. If the GPU threads were to execute the double count removal simultaneously, namely, if the classing method were not used, two rays that have intersected an observation point and represent a double count would not be properly flagged, and the double count would be missed.

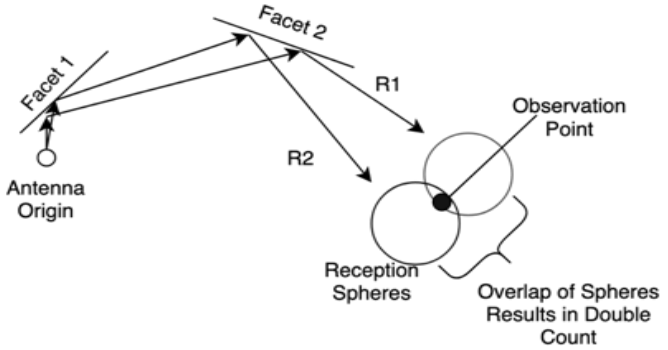

Fig. 2. Ray cone double counted wavefront**:** Illustration of overlapping ray cones, with multiple adjacent cones intersecting the same observation point (the black dot) due to the overlap.

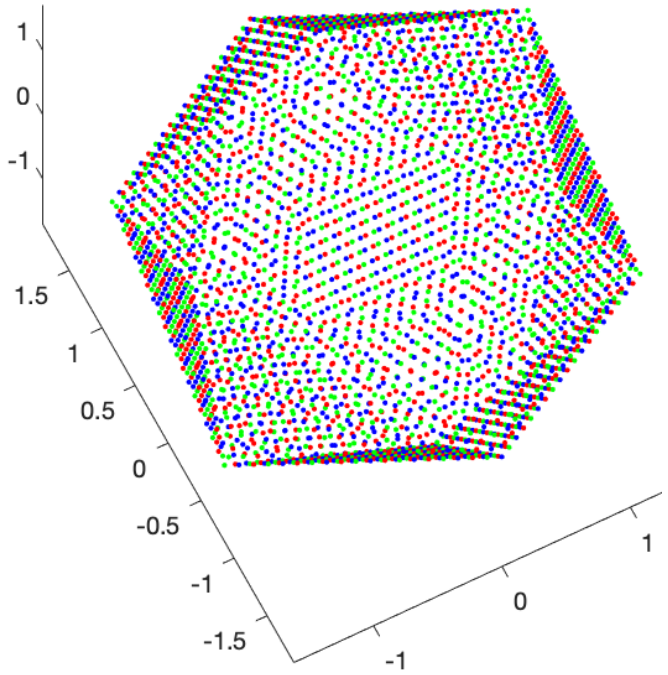

Fig. 3. Adjacent classes of the RT icosahedron**:** The three distinct classes displayed as red, green, and blue dots plotted over the icosahedron display how adjacent rays are members of different classes.

C. SBR Electric Field Computation Acceleration

Typical SBR algorithms use the physical optics (PO) radiation approximation for the electric field computations. Essentially, the rays are used to "paint" surface currents in the environment, which are then entered in the PO integral to compute scattered fields. These integrations can be carried out for each mesh facet individually [7], which does allow for some degree of parallelization. However, the numerical integration is still computationally heavy for GPU implementation, which can result in an efficiency bottleneck. In the method presented here, instead of using the PO approximation, the SBR technique implements a ray-sphere intersection method, as detailed in [3]. The advantage is that each ray-sphere test is entirely independent from one another,

as well as computationally efficient. This results in an electric field computation scheme that allows for embarrassingly parallel GPU methodology.

Our methodology uses the NVIDIA CUDA API to search ray path intersections in parallel, where every ray is checked for intersection with observation point spheres, which represent the radius of the ray-cone. Since it is possible for a ray to intersect multiple observation points, every ray must be checked with every sphere, which results in a computational complexity of O(*NP*), where *N* is the number of rays and *P* is the number of observation points. Due to the nature of arrays on GPUs, it is impossible to allocate an array in memory without knowledge of the array size, and, since the number of ray-observation intersections cannot be known a priori, ray-observation checks need to be conducted twice to achieve parallelism, as follows. Our technique checks for the ray-observation intersection, and if found true, it uses atomic add to create a total count of intersections, which is then used to allocate an array on the GPU. The same intersection checks are done again, but now we fill the array with electric field data that is used to calculate the total electric field.

While this unavoidable doubling of the ray-observation checks, for GPU parallelism, certainly decreases the efficiency of the analysis per se, given that typically both *N* and *P* are very large in RT applications, and so is O(*NP*), the overall order of magnitude for the computation time remains the same, and hence this method still massively benefits from parallelization. To illustrate this, Fig. 4. shows the comparison of the ray-sphere intersection method time when done on a GPU versus a CPU. In the CPU intersections, C++ vectors are leveraged so that the intersections do not need to be computed twice. Instead, a vector push back is used to extend the container for each valid intersection. However, due to the inefficient nature of vector reallocation, as well as the fully parallel computations on the GPU, we see an increase of up to 125× for just this portion of the code. This is also indicative of a massive increase in computational efficiency of the overall SBR RT methodology with the proposed GPU parallelization, resulting in an unparalleled total computation time relative to other approaches.

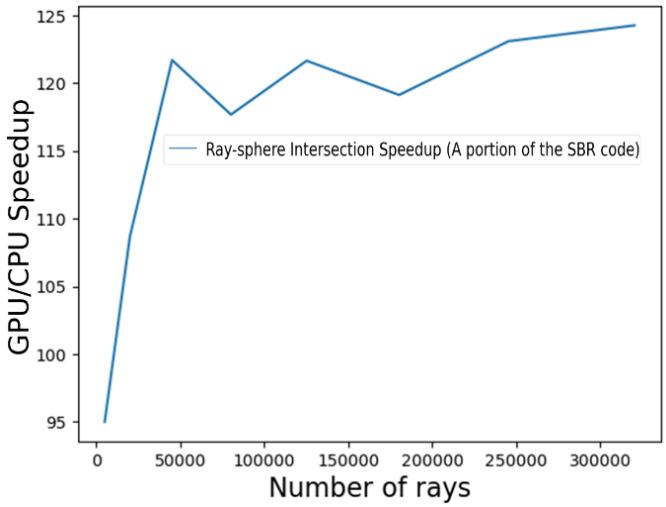

Fig. 4. Speedup of the ray-observation intersection method (only a portion of the overall SBR RT code), vs. the number of rays, as implemented on a GPU in parallel, with two passes of the intersection checks method, relative to the implementation on a CPU in serial, with intersections computed once.

IV. Numerical Results and Discussion

As the first example of the computational speedup from the proposed novel NVIDIA OptiX Prime CUDA GPU parallelization framework for SBR RT CEM propagation modeling, Fig. 5 shows strong scaling results of the SBR code when compared to Amdahl's parallelization scaling law at 99.3% parallelism, when using 2.5 million rays and 20 reflections, which are typical values required for convergence of the SBR method. The number of CUDA blocks is set to 1 while increasing the number of compute threads per block from 1 to 1024, to increase parallel computations, where 1024 threads per block is a limit for this comparison imposed by CUDA. These strong scaling results indicate that the code has achieved about 99% parallelism, namely, a 99% parallel portion with 1% sequential rest. This is an excellent outcome, generally extremely difficult to realize due to many postprocessing tasks being traditionally limited to sequential operation. We also perform the evaluation of the level of parallelism using Amdahl's law specifically for the electric field computation portion of the SBR algorithm and code, along with the parallelism for the entire algorithm and code. The results show very similar parallelism levels, of about 99%, for both the entire code and the field computation part.

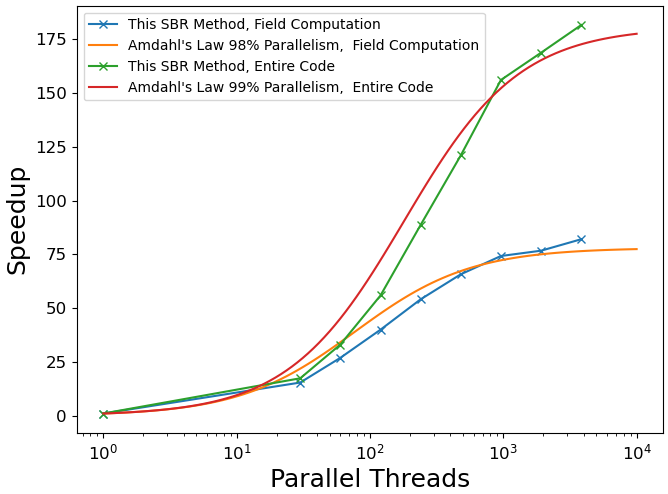

Fig. 5. Comparison of strong scaling SBR RT results with 2.5 million rays and 20 reflections to Amdahl's law with 99% parallelism, curve-fitted through the SBR data points. The GPU computations are performed a variable number of compute threads (per block).

As the next code efficiency demonstration, Fig. 6 shows a comparison with Amdahl's law of strong scaling results obtained with the same accelerated SBR framework as in Fig. 5 but for the number of threads per block set to 128 and the number of CUDA blocks swept from 1 to 4096. We observe from this experiment a strong speedup and approximately 97% parallelism in the code even when a larger number of total compute threads are used. We also see the speedup plateau at around 1000 blocks, which signifies full occupancy

of the GPU (once the memory is at full occupancy, speedup is effectively bound) and numerically confirms related theoretical and empirical considerations.

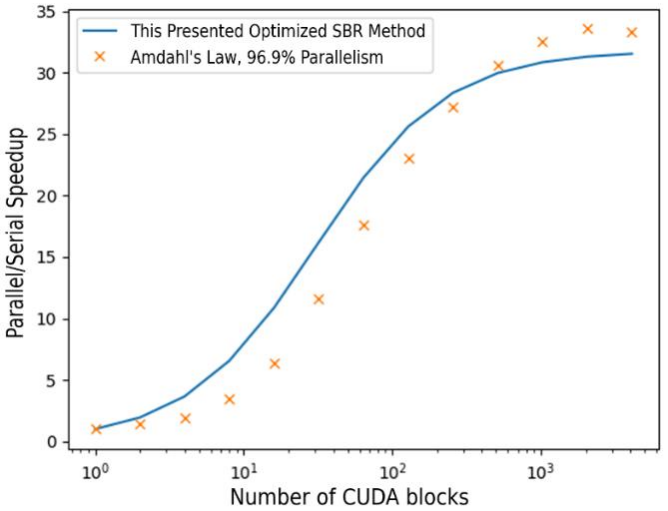

Fig. 6. Comparison of strong scaling results to Amdahl's law with 96.9% parallelism (curve fit through SBR points) for the same accelerated SBR framework as in Fig. 5 but with a fixed number (128) of compute threads per CUDA block and a variable number of blocks.

Next, Table I shows comparison of total computation times of the SBR RT propagation analysis conducted in serial and parallel, that is, for the results generated using a serial CPU implementation and with the parallelized GPU code, respectively. The simulations are performed using optimal values for compute threads and CUDA blocks for individual numbers of rays, on a standard workstation computer with an NVIDIA Geforce 1060 (6 GB) GPU. We observe massive parallel vs. serial speedups, which demonstrates the effectiveness of the proposed methodology in truly optimizing the SBR RT code and taking full advantage of NVIDIA OptiX Prime API and CUDA GPU programing interface for propagation modeling.

TABLE I. COMPARISON OF SERIAL AND PARALLEL COMPUTATION TIMES FOR SBR RT PROPAGATION COMPUTATION.

Number of Rays (thousands)	100	400	900	1600
Serial Computation Time (seconds)	852	6181	29638	86214
Parallel Computation Time (seconds)	1	2.6	4.5	6.5
Speedup (×)	852	2372	6565	13263

As the next example, Fig. 7 shows propagation modeling results for a rectangular dielectric waveguide (tunnel) from [5], with dimensions 4 m × 4 m × 1000 m, tunnel wall dielectric parameters $\varepsilon_r = 5$ and $\sigma = 0$, and excitation by a vertically polarized isotropic transmitter at 1 GHz. Tunnels represent a challenging environment for RT solvers, due to the large distances covered and the higher order modes present in propagation, and thus an excellent test case. The SBR RT results are compared with the IT RT solution (image theory is known to produce the most accurate ray tracing results) and the results obtained by Ansys Savant SBR+ [18], which uses the PO approximation for electric field computations and is parallel compute enabled. We observe excellent agreement of the results obtained by the three methods. This confirms that the vast parallel optimization of all computational aspects of the SBR RT methodology proposed and described in this work, enabling such high parallelism and unparalleled efficiency, have not affected the previously advanced and established accuracy of the method (on par with the drastically less efficient IT RT approach) [3]. In fact, the presented major efficiency optimizations and enhancements have been enabled by and addressed synergistically with the improvements of the accuracy of the SBR computation.

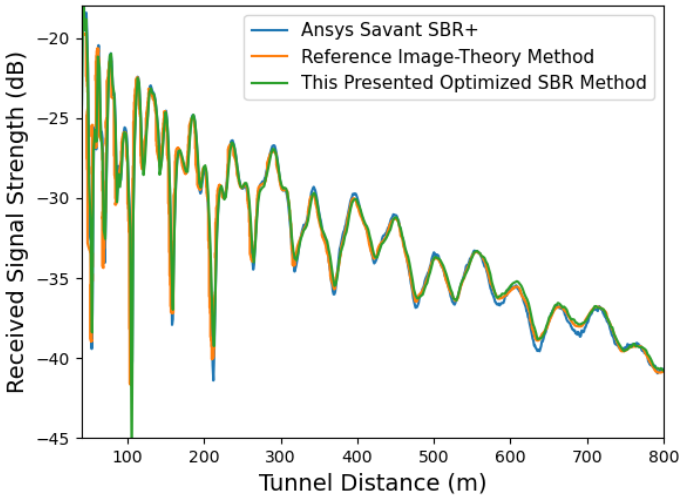

Fig. 7. Comparison of the presented optimally parallelized SBR method with IT [5] and Ansys Savant SBR+ [18] in a 4 m × 4 m × 1000 m waveguide tunnel with dielectric ($\varepsilon_r = 5$, $\sigma = 0$) walls at 1 GHz [transmitter at (1.1 m, 2.1 m, 0), receivers at (1.9 m, 1.7 m, z)].

As the final test case, we consider an example of an urban wireless propagation environment, the so-called "Madrid grid" urban scenario, often referenced as the METIS project scenario, which has been commonly used for evaluation and characterization of wireless propagation and parametrization of radio channel modeling in outdoor urban scenarios, typically via ray-tracing simulations [19]-[22]. This test case is aimed at evaluation and demonstration of both the parallelism of the new proposed optimized SBR methodology and the accuracy and validity of the presented SBR computations. We consider the geometry shown in Fig. 8, in which both the buildings and ground plane are assumed to be concrete, with permittivity of 4.5 and conductivity of 0.09 S/m respectively. The heights of the buildings are adopted to be 50 m. The path from A to B to C represents a line-of-sight radio channel while the path B to D is a non-line-of-sight channel. The transmitting antenna (Tx), at location A, is a vertically polarized half-wave dipole at a frequency of 5 GHz, at a height of 10 m with respect to the ground. The receiving points are placed at heights of 5 m. Note that the presented SBR methodology is a pure SBR RT approach, and hence diffraction from edges is not included in the model.

This article has been accepted for publication in IEEE Transactions on Antennas and Propagation. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TAP.2024.3350896

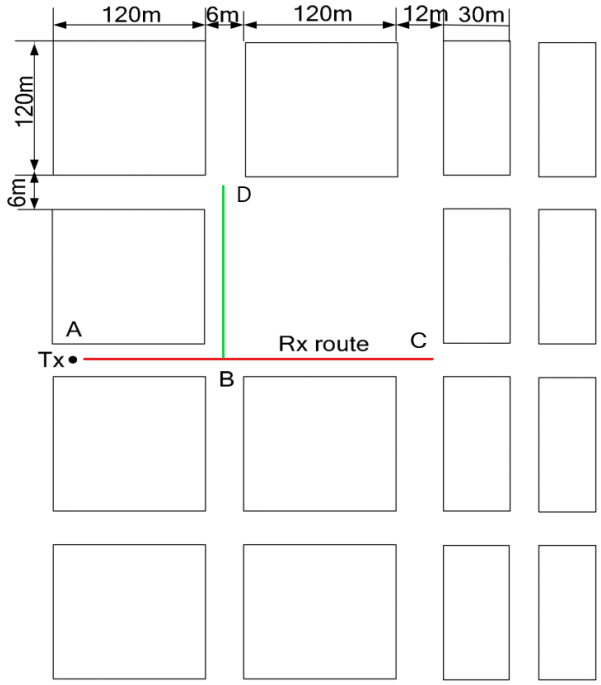

Fig. 8. Geometry of the "Madrid grid" (METIS project) outdoor urban scenario. The path A–B–C represents a line-of-sight radio channel while the path B–D is a non-line-of-sight channel. The transmitting antenna (Tx) at 5 GHz is at location A.

The validation of our SBR computations for this urban radio channel modeling scenario is performed using Ansys SBR+ as shown in Fig. 9. We observe excellent agreement of the present SBR method with Ansys SBR+ in both the line-of-sight and non-line-of-sight channels.

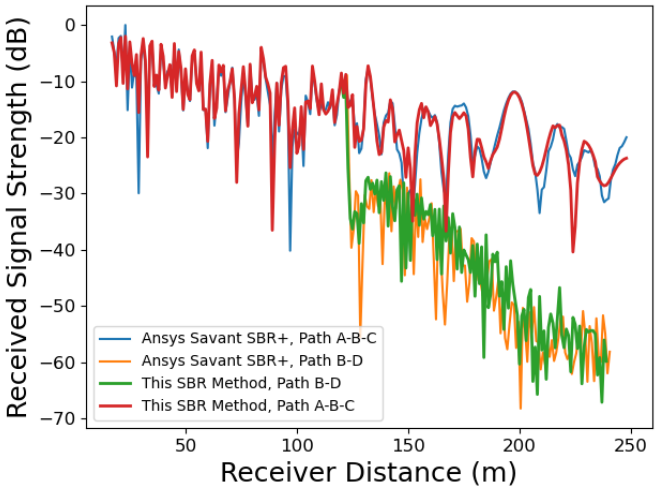

Fig. 9. Results for the METIS test case geometry from Fig. 8. A vertically polarized half-wave dipole is launched from Tx and traced through the Rx path shown at 5 GHz. The buildings and ground are both assumed to be cement with a relative permittivity of 4.5 and conductivity of 0.09 S/m.

While showing this excellent agreement of the results, we also demonstrate massive parallelization for this test case. The results in Fig. 10 show very high parallelism levels, namely, upwards of 99% parallelism, for both the entire code and the field computation part in the "Madrid grid" urban scenario SBR simulations, similar to the tunnel test case.

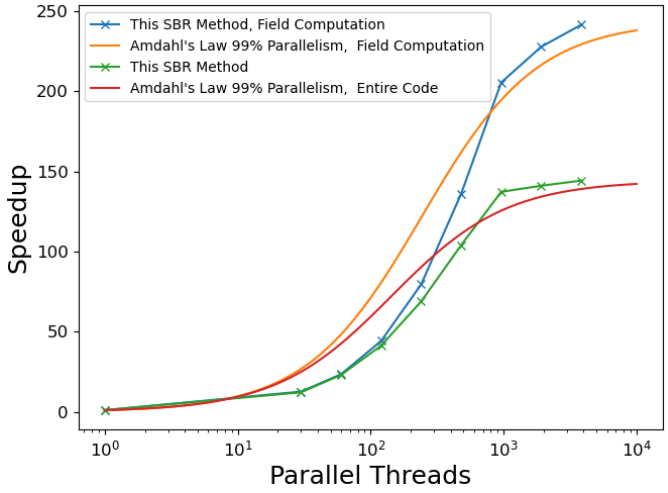

Fig. 10. Strong scaling for the METIS test case. We show scaling for the electric field computation portion of the code as well as the total scaling for the SBR method (the entire code).

VI. Conclusion

This paper has proposed and described a novel parallelization methodology to maximally enhance the efficiency of the developed cone-based shooting and bouncing ray tracing technique for electromagnetic propagation modeling, which incorporates procedures such as ray batching and double count removal to improve the accuracy of the solution. The parallel optimization of the SBR code is achieved by fully exploiting the inherent features of NVIDIA OptiX Prime API and CUDA GPU programing interfaces suitable for SBR computations, comprehensive parallelization of all components of the SBR algorithm, including electric field calculation as well as postprocessing tasks being traditionally limited to sequential operation, and addressing and optimizing memory usage and constraints to further advance efficiency of the method. The paper has presented and explained in detail a novel unified framework of algorithms, strategies, and data structures applied to SBR RT to further acceleration on the GPU without sacrificing the previously advanced and established accuracy of the method. Importantly, the major enhancements of the accuracy and the efficiency are tightly coupled and had to be addressed and pursued synergistically.

The presented numerical results have demonstrated that the new proposed optimized SBR methodology achieves massive parallel vs. serial speedups and upwards of 99% parallelism under Amdahl's parallelization scaling law, which is a rather unique performance outcome. The strategic use of GPUs and the meticulous optimization of all computational aspects of the code have resulted in a solver that is extremely efficient in launching millions of rays, computing, storing, and updating the electric field for each ray, and converging to accurate solutions for electrically large structures smoothly and rapidly. The efficiency gained from the acceleration, achieved

from the very high level of parallelism, has created an SBR algorithm with a favorable performance and dramatic reduction of computation time for convergence in long tunnels, as a challenging test case, as well as in an urban outdoor wireless propagation environment. We believe that the algorithms, strategies, and data structures proposed and described in this work should be of significant interest and value to researchers, developers, and practitioners in the field of ray-based and high-frequency asymptotic methods and modeling, as well as for all related considerations and advancements of parallelism and efficiency of CEM techniques in general.

REFERENCES

[1] M.F.Cátedra and J.Perez, Cell Planning for Wireless Communications. Norwood, MA, USA: Artech House, 1999.

[2] S.-H. Chen and S.-K. Jeng, "SBR image approach for radio wave propagation in tunnels with and without traffic," *IEEE Transactions on Vehicular Technology*, vol. 45, no. 3, Aug. 1996, pp. 570-578.

[3] S. Kasdorf, B. Troksa, C. Key, J. Harmon, and B. M. Notaros, "Advancing Accuracy of Shooting and Bouncing Rays Method for Ray-Tracing Propagation Modeling Based on Novel Approaches to Ray Cone Angle Calculation," *IEEE Transactions on Antennas and Propagation*, vol. 69, no. 8, pp. 4808–4815, August 2021.

[4] H. Ling, R.-C. Chou, and S.-W. Lee, "Shooting and bouncing rays: calculating the RCS of an arbitrarily shaped cavity," *IEEE Transactions on Antennas and Propagation*, vol. 37, no. 2, pp. 194-205, Feb. 1989.

[5] D. Didascalou, "Ray Optical Wave Propagation Modelling in Arbitrarily Shaped Tunnels," Ph.D. dissertation, Dept. Elect. Eng., *Universität Karlsruhe*, Karlsruhe, Germany, 2000.

[6] B. M. Notaros and L. C. Kempel, "Computational Electromagnetics for Antennas," in *Antenna Engineering Handbook, 5th edition*, J. L. Volakis (ed.), *McGraw-Hill Education*, pp. 1343–1375, 2018.

[7] C. Y. Kee and C. Wang, "Efficient GPU Implementation of the High-Frequency SBR-PO Method," *IEEE Antennas and Wireless Propagation Letters*, vol. 12, pp. 941-944, 2013.

[8] K. Xu, D. Ding and R. Chen, "Programmable graphics processing units (GPUs) accelerated SBR method for analyzing the scattering of open cavities," *2008 Asia-Pacific Microwave Conference*, pp. 1-4, 2008.

[9] C. Key, B. A. Troksa, S. Kasdorf, and B. M. Notaros, "Non-Self-Adjacent Ray Classes for Parallelizable Shooting–Bouncing Ray Tracing Double Count Removal," *IEEE Journal on Multiscale and Multiphysics Computational Techniques*, vol. 5, pp. 245-254, 2020.

[10] B. A. Troksa, "GPU Accelerated Cone Based Shooting Bouncing Ray Tracing," M.S. Thesis, Department of Electrical and Computer Engineering, Colorado State University, 2019.

[11] S. Kasdorf, B. Troksa, J. Harmon, C. Key, and B. M. Notaros, "Shooting-Bouncing-Rays Technique to Model Mine Tunnels: Algorithm Acceleration," invited paper, Special Session "High Performance Computing in Electromagnetics," *Proceedings of the 2020 International Applied Computational Electromagnetics Society (ACES) Symposium – ACES2020*, Online Conference, July 27–31, 2020.

[12] S. G. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison, and M. Stich, "OptiX: a general purpose ray tracing engine," *ACM Transactions on Graphics*, vol. 29, no. 4, pp. 1-13, 2010.

[13] A. Santos, J. M. Teixeira, T. Farias, V. Teichrieb, and J. Kelner, "Understanding the Efficiency of kD-tree Ray-Traversal Techniques over a GPGPU Architecture," *International Journal of Parallel Programming*, vol. 40, no. 3, pp. 331–352, 2011.

[14] C Woolley, "GPU optimization fundamentals," NVIDIA Corp., Dev. Tech. Eng., 2013.

[15] N. Noori, A. A. Shishegar and E. Jedari, "A New Double Counting Cancellation Technique for Three-Dimensional Ray Launching Method," *2006 IEEE Antennas and Propagation Society International Symposium*, Albuquerque, NM, 2006, pp. 2185-2188.

[16] Z. Yun, M. F. Iskander, and Z. Zhang, "Development of a new shooting-and-bouncing ray (SBR) tracing method that avoids ray double counting," *IEEE Antennas and Propagation Society International Symposium. 2001 Digest*. Boston, MA, USA, pp. 464-467, 2001.

[17] M. F. Iskander and Z. Yun, "Propagation prediction models for wireless communication systems," *IEEE Transactions on Microwave Theory and Techniques*, vol. 50, no. 3, pp. 662-673, March 2002.

[18] Ansys Savant 2023 R1, ANSYS, Inc. Available: http://www.ansys.com. Accessed June 28, 2023.

[19] J. F. Monserrat and M. Fallgren, *METIS Deliverable D6.1 Simulation guidelines[J]*, October 2013, [online] Available: http://ww.metis2020.com/.

[20] A. W. Mbugua, Y. Chen, L. Raschkowski, L. Thiele, S. Jaeckel and W. Fan, "Review on Ray Tracing Channel Simulation Accuracy in Sub-6 GHz Outdoor Deployment Scenarios," in *IEEE Open Journal of Antennas and Propagation*, vol. 2, pp. 22-37, 2021

[21] G. Steinböck, A. Karstensen, P. Kyösti and A. Hekkala, "A 5G hybrid channel model considering rays and geometric stochastic propagation graph," *2016 IEEE 27th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, Valencia, Spain, 2016, pp. 1-6

[22] C. Feng and L. Yuan-jian, "Study on propagation characteristics for typical outdoor environment," *2015 Asia-Pacific Microwave Conference (APMC)*, Nanjing, China, 2015, pp. 1-3.

Stephen Kasdorf (Student Member, IEEE) received B.S. (magna cum laude) degrees in 2019 in both electrical engineering and applied physics from Colorado State University. He is currently working towards a PhD in electrical engineering at Colorado State University. His research interests include high frequency asymptotic electromagnetics methods such as ray optics and physical optics, as well as full-wave numerical techniques, uncertainty quantification, surrogate models, and adjoint solutions.

Blake Troksa (Student Member, IEEE) was born in Boulder, CO in 1996. He received his B.S. (2018) and his M.S. (2019) in Electrical and Computer Engineering from Colorado State University. He is currently working as a software development engineer in the area of cloud computing. His research interests include hardware acceleration and distributed systems.

Cam Key (Student Member, IEEE) was born in Fort Collins, CO in 1996. He received his B.S. (2018) and his Ph.D. (2020) in Electrical and Computer Engineering from Colorado State University. His current research interests include uncertainty quantification, error prediction, and optimization for computational science and engineering; computational geometry, meshing, data science, machine learning, artificial intelligence, remote sensing and GIS, and novel applications of numerical methods across disciplines.

Jake Harmon (Member, IEEE) received the B.S. degree (summa cum laude) in 2019, graduating top of his class, and the PhD degree in 2022, both from Colorado State University, Fort Collins, CO, USA, in Electrical Engineering. His PhD research, which was supported by the DoD High Performance Computing and Modernization Program and the US Air Force Research Laboratory (AFRL), was on the topics of adaptive numerical methods and uncertainty quantification, including goal-oriented error estimation and adaptivity, hp-refinement, and

expediting the propagation of uncertainty in computational electromagnetics. His research interests include adaptive numerical methods, uncertainty quantification, and scientific computing.

Sudeep Pasricha (Fellow, IEEE), received his Ph.D. in Computer Science from the University of California, Irvine in 2008. He is currently a Walter Scott Jr. College of Engineering Professor in the Department of Electrical and Computer Engineering, at Colorado State University. His research focuses on the design of innovative software algorithms, hardware architectures, and hardware-software co-design techniques for energy-efficient, fault-tolerant, real-time, and secure computing. He has co-authored five books and published more than 300 research articles in peer-reviewed journals and conferences that have received 17 Best Paper Awards and Nominations at various IEEE and ACM conferences. He has served as General Chair and Program Committee Chair for multiple IEEE and ACM conferences and served in the Editorial board of multiple IEEE and ACM journals. He is a Fellow of the IEEE, a Distinguished Member of the ACM, and an ACM Distinguished Speaker.

Branislav M. Notaroš (Fellow, IEEE) received the Dipl.Ing. (B.S.), M.S., and Ph.D. degrees in electrical engineering from the University of Belgrade, Belgrade, Yugoslavia, in 1988, 1992, and 1995, respectively. From 1996 to 1999, he was Assistant Professor in the School of Electrical Engineering at the University of Belgrade. He was Assistant and Associate Professor from 1999 to 2006 in the Department of Electrical and Computer Engineering at the University of Massachusetts Dartmouth. He is currently Professor of Electrical and Computer Engineering, University Distinguished Teaching Scholar, and Director of Electromagnetics Laboratory at Colorado State University. His research contributions are in computational and applied electromagnetics. His publications include more than 300 journal and conference papers, and textbooks "Electromagnetics" (2010) and "MATLAB-Based Electromagnetics" (2013) with Pearson Prentice Hall and "Conceptual Electromagnetics" (2017) with CRC Press.

Prof. Notaroš serves as President of the IEEE Antennas and Propagation Society (AP-S), Immediate Past President of the Applied Computational Electromagnetics Society (ACES), Chair of the USNC-URSI Commission B, and Track Editor of the IEEE Transactions on Antennas and Propagation. He served as General Chair of the IEEE APS/URSI 2022 Denver Conference, Chair of the IEEE AP-S Meetings Committee, Chair of the Joint Meetings Committee, and AP-S AdCom member. He was the recipient of the 1999 IEE Marconi Premium, 2005 IEEE MTT-S Microwave Prize, 2022 IEEE Antennas and Propagation Edward E. Altshuler Prize Paper Award, 2019 ACES Technical Achievement Award, 2014 Carnegie Foundation Colorado Professor of the Year Award, 2015 ASEE ECE Distinguished Educator Award, 2015 IEEE Undergraduate Teaching Award, and many other research and teaching awards. He is Fellow of IEEE and ACES.